intel

APPLICATION NOTE

AP-123

March 1982

Graphic CRT Design Using the Intel® 8089

Hal Kop
Microcomputer Applications

MARCH 1982
ORDER NUMBER: 210355-001

Graphic CRT Design Using the Intel 8089

Contents

INTRODUCTION

The purpose of this application note is to provide the reader with the conceptual tools and factual information needed to apply the Intel 8089 to graphic CRT design. Particular attention will be paid to the requirements of high-resolution, color graphic applications, since these tend to require higher performance than those which do not use color.

The Intel 8089 is a microprocessor system which contains an 8086 CPU and an 8089 Input/Output Processor. In the graphic CRT application, the 8089 performs DMA transfers from the display memory to the CRT controller, and also serves as a CPU for functions such as keyboard polling and initialization of the CRT controller chips. The DMA transfers are done in such a manner that they do not tie up the system bus.

The system is organized so that the 8086 and the 8089 can perform concurrent processing on separate buses. Using the inherent ability of the 8089 to execute programs in its own I/O space, the 8086 can successfully delegate many of the chores that have specifically to do with the CRT display and keyboard, thus reducing the 8086's processing overhead. For these reasons, the capabilities of the 8086 as a CPU can be more fully utilized to perform calculations dealing with the material to be displayed. Thus, more complex types of displays can be undertaken, and the terminal will also be more interactive.

This application note is presented in five sections:

1. Introduction
2. Overview of Graphic CRT Systems
3. Overview of the 8089
4. Graphic CRT System Design
5. Conclusions

Section 2 discusses typical CRT designs, shows how performance requirements increase when the capability for color graphics is included, and explains some of the system bottlenecks that can arise. Section 3 describes the capabilities of the 8089, which can be brought to bear to resolve these bottlenecks. Section 4 gives detailed information for a color graphic CRT system using the Intel 8089 (8086 and 8089).

The reader may obtain useful background information on the 8086 and 8089 from *iAPX 86,88 User's Manual*. It would also be helpful to read the data sheets on the 8086, 8089, 2118 Dynamic RAM, 8202 Dynamic Ram Controller, 8275 CRT Controller, 8279 Keyboard/Display Interface, and 2732A EPROM.

OVERVIEW OF CRT GRAPHIC SYSTEMS

Typical Design Technique

A typical microprocessor-based CRT terminal is shown in block diagram form in Figure 1. The terminal consists

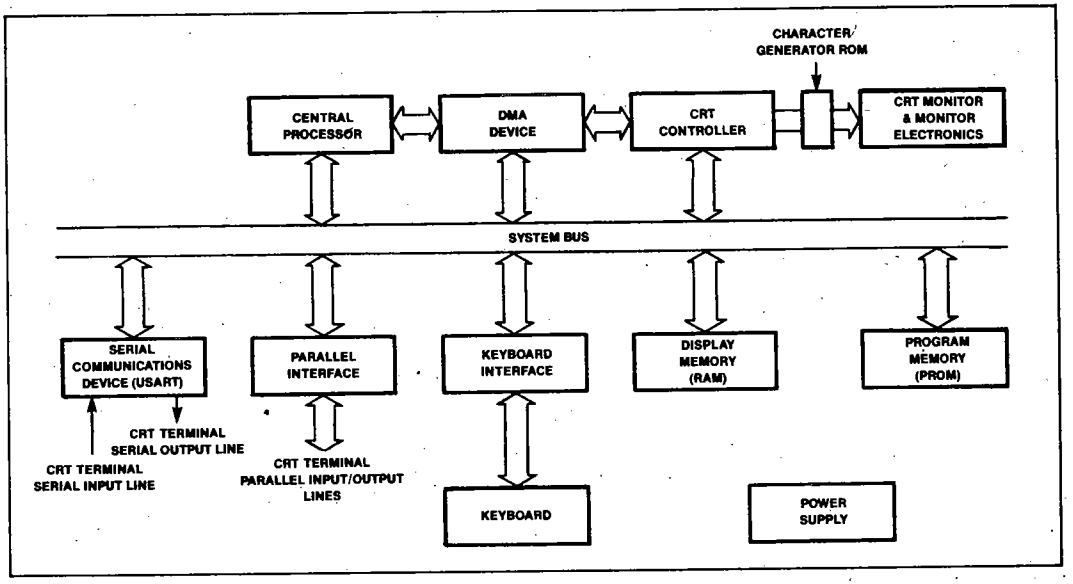


Figure 1. Typical CRT Terminal Block Diagram

of a CRT monitor, monitor electronics, a CRT controller and character generator ROM, display memory, a DMA device, a central processor and associated program memory, a keyboard and keyboard interface, and serial and/or parallel communication devices.

The primary function of the non-graphic CRT controller is to refresh the display. It does this by controlling the periodic transfer of information from display memory to the CRT screen, with the help of the DMA device. The central processing unit (CPU) coordinates the transfer of information to and from the external devices. When information from an external device is received by the terminal, the CPU performs character recognition and handling functions, display memory management functions, and cursor control functions. The CPU also interrogates the keyboard interface device. If a key depression is detected, the ASCII character representing that key is sent to the display memory and/or an external device.

The design shown in Figure 1 could be implemented using Intel LSI products. The CPU could be an 8085, the DMA device an 8237A DMA controller, the CRT controller an 8275, the character generator ROM a 2708, program memory ROM a 2716, display memory 2114s (2K x 8), and the keyboard interface an 8279 keyboard controller. These choices would result in a CRT terminal capable of displaying 25 lines of text containing 80 characters each.

As the design is upgraded to add color and graphics capability, performance requirements increase accordingly. The components most likely to require changing are the CPU, the DMA device, the CRT controller, and the display memory. Thus, it is desirable at this point to examine the operation of these components in more detail to provide a foundation for graphic system operation. Later we shall give a specific example of a more complex display, and examine the performance requirements imposed. Figure 2 is a block diagram showing only those components involved with the non-graphic CRT refresh function, with more detail provided regarding the connecting signal lines.

The refresh function proceeds as follows. The 8275, having been programmed to the specific screen format, generates a series of DMA request signals to the 8237A. This results in the transfer of a row of characters from display memory to one of two row buffers within the 8275. From this row buffer, the characters are sent, via lines CC0–CC6, to the character generator ROM. The dot timing and interface circuitry is then utilized to convert the parallel output data from the character generator ROM into serial signals for the video input of the CRT.

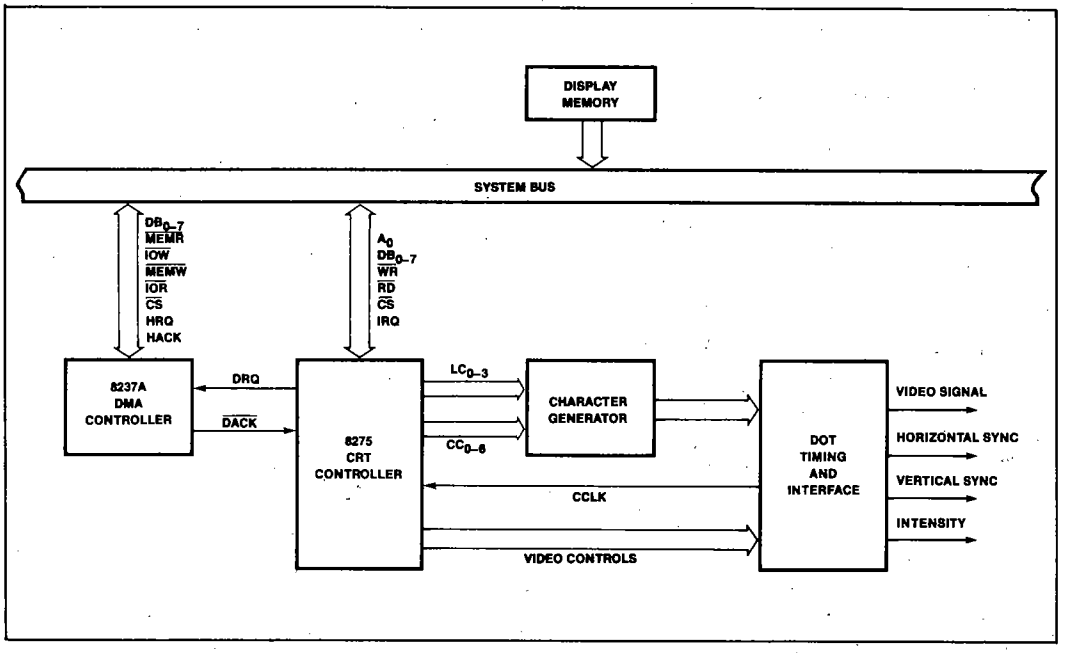


Figure 2. Components Involved in the CRT Refresh Function

210355-001

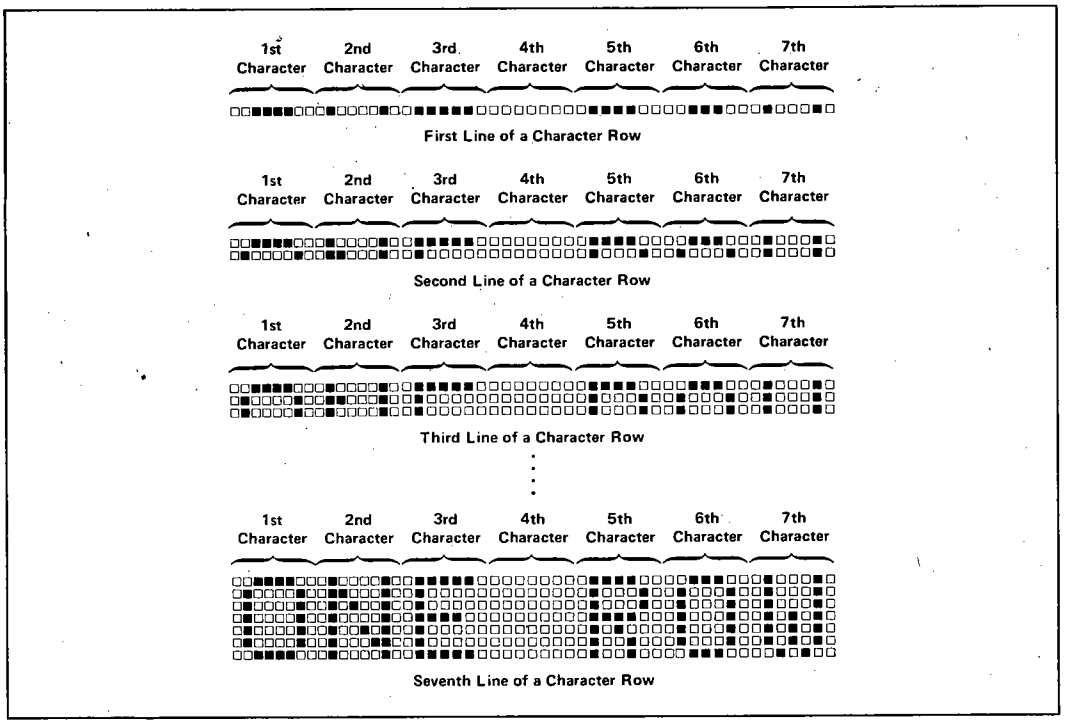


Figure 3. Character Row Display

The character rows are displayed on the CRT one line at a time. Line count signals LC0–LC3 are applied to the character generator ROM by the 8275, to specify the specific line count within the row of characters. This display process is shown in Figure 3, using a seven-line character for purposes of illustration. The entire process is repeated for each row of characters in the display.

At the beginning of the last display row, the 8275 issues an interrupt request via the IRQ output line. This interrupt output is normally connected to the interrupt input of the system CPU. The interrupt causes the CPU to execute an interrupt service subroutine. This subroutine typically reinitializes the DMA controller parameters for the next display refresh cycle, polls the system keyboard controller, and executes other appropriate functions.

Performance Requirements

In the example we have discussed thus far, a display consisting of 25 rows, each containing 80 text characters, with no color or graphic capability, has been assumed. Such a screen can be represented by 80 x 25 = 2000 bytes of data. If the screen is refreshed 60 times per second, then a total of 120,000 bytes will need to be transferred each second from display memory to the 8275 CRT controller. This figure is well within the capability of the 8237A DMA controller, even allowing for vertical retrace time and other overhead. In this application then, both the display memory and the system bus remain available to the system CPU most of the time, and no bottleneck occurs because of the DMA transfer process.

The situation is quite different when a high-resolution, color graphics capability is desired. The performance requirements are obviously much greater. To derive a quantitative requirement it is necessary to choose, even if somewhat arbitrarily, a specific display method and screen format. The display method chosen for the system described in this application note is called the virtual-bit mapping technique. When this technique is used, the graphic material to be displayed is handled on a character basis. Figure 4 shows the structure of the text and graphic characters used. The text character is a

7 x 5 character in an 8 x 5 matrix. The graphic character is a 4 x 5 matrix.

The size of a graphic character is the same as the size of a text character. In addition, the text characters may be in color. The resolution (horizontal) for a graphic character is twice as coarse as the dot spacing for a text character. One of eight colors may be selected for foreground and for background within a particular character.

Figure 5 shows how the display character can be specified using four bytes. The first byte determines whether the character is a text character or a graphic character, and specifies the colors for foreground and background. If it is a text character, the second byte specifies the character with a seven-bit ASCII code, and the remaining two bytes are not used. If it is a graphics character, the second, third, and fourth bytes contain the color specification for each of the twenty distinct picture elements (pixels) within the character. Use of the foreground color is indicated by a one in the respective bit position, while a zero specifies use of the background color.

The screen format chosen has 80 characters per row and 48 rows. Thus the resolution (in terms of picture elements) is 640 x 480 for text characters and 320 x 240 for graphic characters. A full screen contains 80 x 48 = 3840 characters. Thus, a single frame of the display can be represented by 3840 x 4 = 15,360 bytes. If the screen were updated 60 times per second, the CRT refresh function would require a DMA transfer rate of 15,360 x 60 = 921,600 bytes per second.

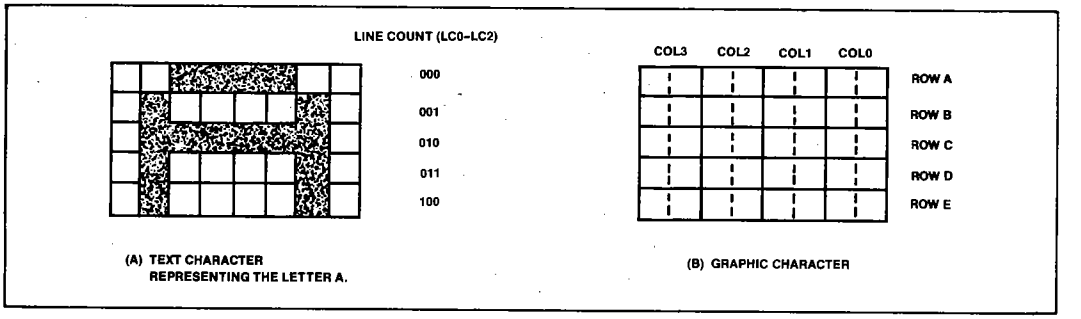


Figure 4. Character Structure

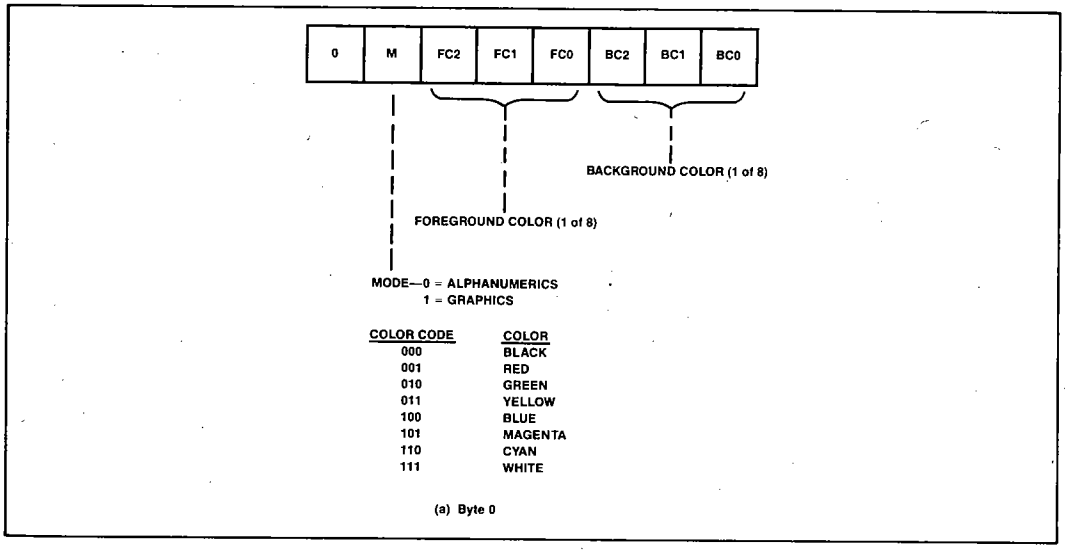
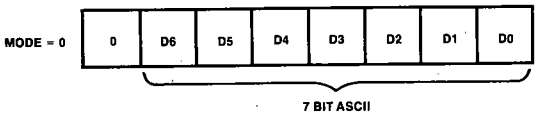
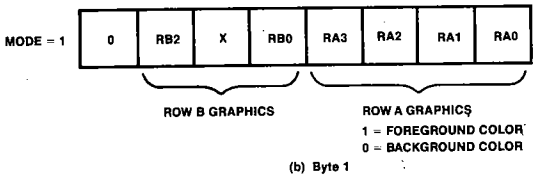


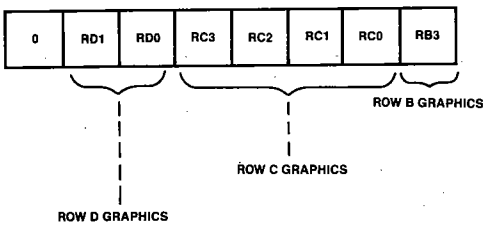
Figure 5. Display Character Specification



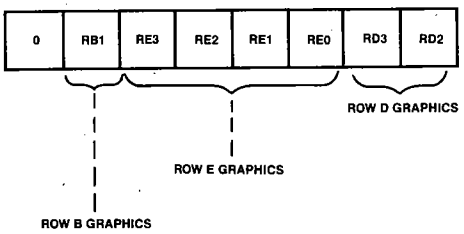


(b) Byte 1

NOTE: RB1 IS INTENTIONALLY MOVED TO BYTE 3 SUCH THAT REPRESENTATION OF A BLANK CHARACTER FOR EITHER TEXT OR GRAPHIC IS THE SAME.



(c) Byte 2



(d) Byte 3

Figure 5. Display Character Specification (Cont.)

System Bottlenecks

It can be seen from the above calculation that nearly one megabyte of data must be transferred per second to effect the CRT refresh function alone. Even with the fastest available DMA controllers, this represents the major part of the bandwidth for such devices. When the design shown in Figure 1 is used, the system bus must also be used by the CRT terminal processor for such functions as keyboard polling and communication with external devices. In addition, any changes made to the material being displayed would require use of the system bus for the purpose of storing the new material in the display memory, and possibly also for access to system memory during the calculation process. It is easy to see, therefore, that severe bottlenecks can occur in terms of system bus utilization. Problems involving bus contention could also be difficult to resolve. Display underruns could become difficult or impossible to avoid in some cases, such as when graphics computations require excessive use of the system bus.

The situation can be improved substantially if provision is made for concurrent processing. One CPU can be doing calculations on the material to be displayed, while another CPU can be managing the CRT terminal functions and the I/O devices simultaneously. Local buses can be used for access to the respective program memories, with the system bus used only for transfer of new display data and for communication between the two processors.

The Intel 8089 offers a convenient and economical way of implementing this multiprocessing approach. In particular, the 8089 has unique capabilities that simplify the design process.

OVERVIEW OF THE 8089

Architectural Overview

The 8089 Input/Output Processor is a complete I/O management system on a single chip. It contains two independent I/O channels, each of which has the capabilities of a CPU combined with a programmable DMA controller.

The DMA functions are somewhat more flexible than those of most DMA controllers. For example, a conventional DMA controller transfers data between an I/O device and a memory. The 8089 DMA function can operate between one memory and another, between a memory and an I/O device, or between one I/O device and another. Any device (I/O or memory) can physically reside on the system bus or on the I/O bus. The bus width for the source and destination need not be the same. If the source, for example, is a 16-bit device, while the destination is an 8-bit device, the 8089 will disassemble the 16-bit word automatically as part of the DMA transfer process. The transfer can be synchronized by the source, by the destination, or it can be free running. The 8089 can effect data transfers at rates up to 1.25 megabytes when a 5 MHz clock is used.

Unlike most DMA controllers, the 8089 uses a two-cycle approach to DMA transfer. A fetch cycle reads the data from the source into the 8089, and a store cycle writes the data from the 8089 to the destination. This two-cycle approach enables the 8089 to perform operations on the data being transferred. Typical of such operations are translating bytes from one code to another (for example, EBCDIC to ASCII) or comparing data bytes to a search value.

A variety of conditions can be specified for terminating DMA transfers, including single cycle, byte count (up to 64K), external event, and data-dependent conditions, such as the outcome of a masked compare operation.

The CPU in each channel can execute programs in the system space (from a memory on the system bus) or in the I/O space (from a memory on a separate I/O bus). Thus, complete channel programs can be run by the 8089 without tying up the system bus or interfering with the operation of the system CPU. Figure 6 is a simplified block diagram of the 8089, showing how the 8089 interfaces with these two buses.

The programs that the 8089 executes may be preexisting programs stored in ROM or EPROM, or they may be programs prepared for the 8089 by the system CPU. In the latter case, the programs are typically in modular form, contained in "task blocks" that the system CPU places in a memory location accessible to the 8089. During normal operation, the system CPU then directs the 8089 to the various task blocks, according to which programs are to be executed. The details of how this is done are given below under *Software Interface*.

The 8089 has an addressing capability of 64K bytes in the I/O space, and thus can support multiple peripherals, as illustrated in Figure 7. In the system space, the 8089 supports 1-megabyte addressing, and is directly compatible with the 8086 or 8088, and with Intel's Multibus. The 8089 operates from a single +5V power source, and is housed in a standard 40-pin, dual in-line package. The instruction set for the 8089 IOP is specifically designed and optimized for I/O processing and control. In addition to being able to execute DMA

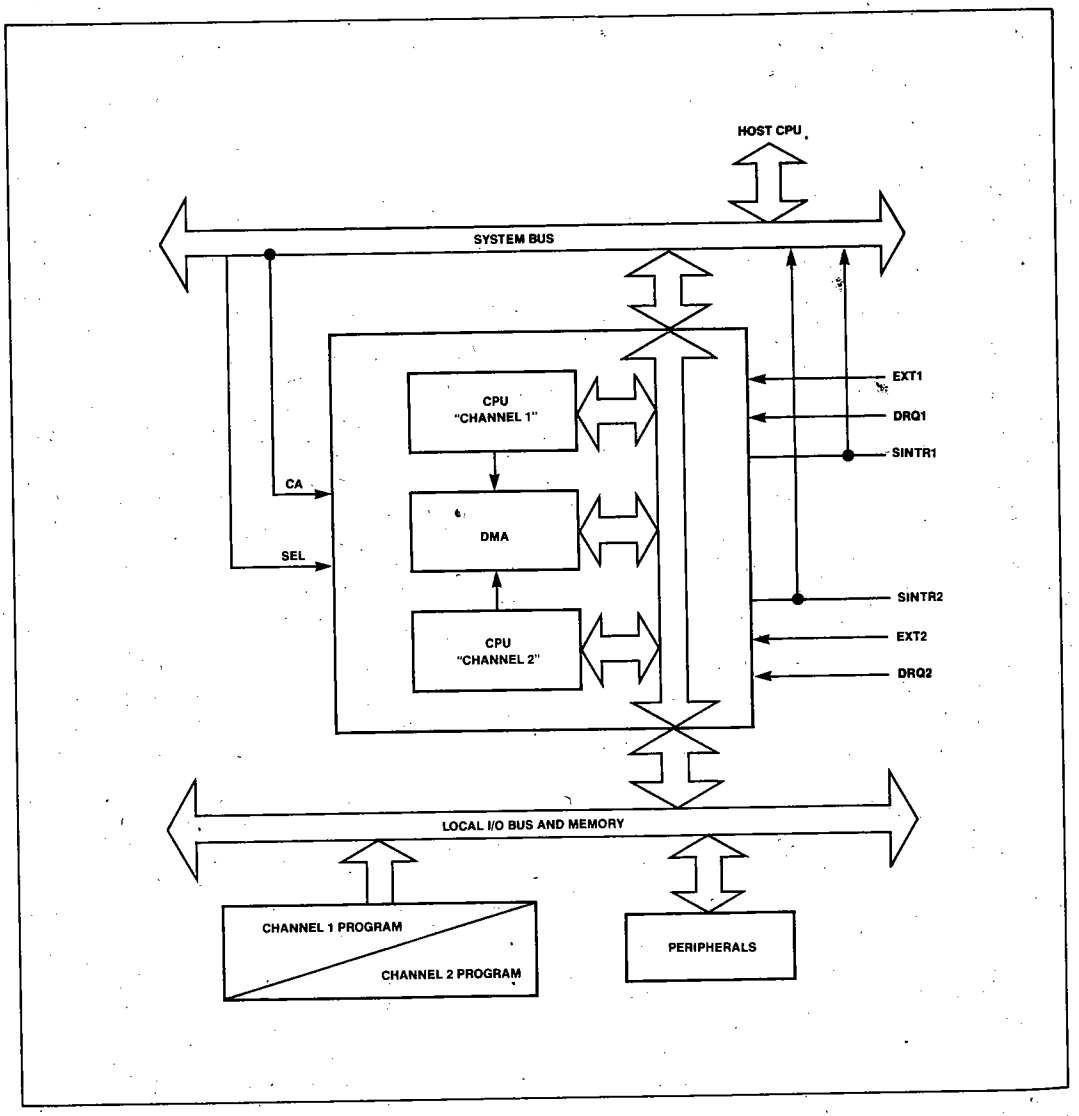


Figure 6. Simplified Block Diagram of the 8089

transfers under a wide variety of operating conditions, the 8089 can perform logic operations, bit manipulations, and elementary arithmetic operations on the data being transferred. A variety of addressing modes may be used, including register indirect, index auto increment, immediate offset, immediate literal, and indexed.

The register set for the 8089 is shown in Figure 8. Each channel has an independent set of these registers, not accessible to the other channel. Table 1 gives a brief summary of how these registers are used during a program execution or during a DMA transfer. Four of the registers can contain memory addresses which refer to either the system space or the I/O space. These registers each have an associated tag bit. Tag = 0 refers to the system space and tag = 1 refers to the I/O space. More details on how the registers are used are given below as part of the *Software Interface* section.

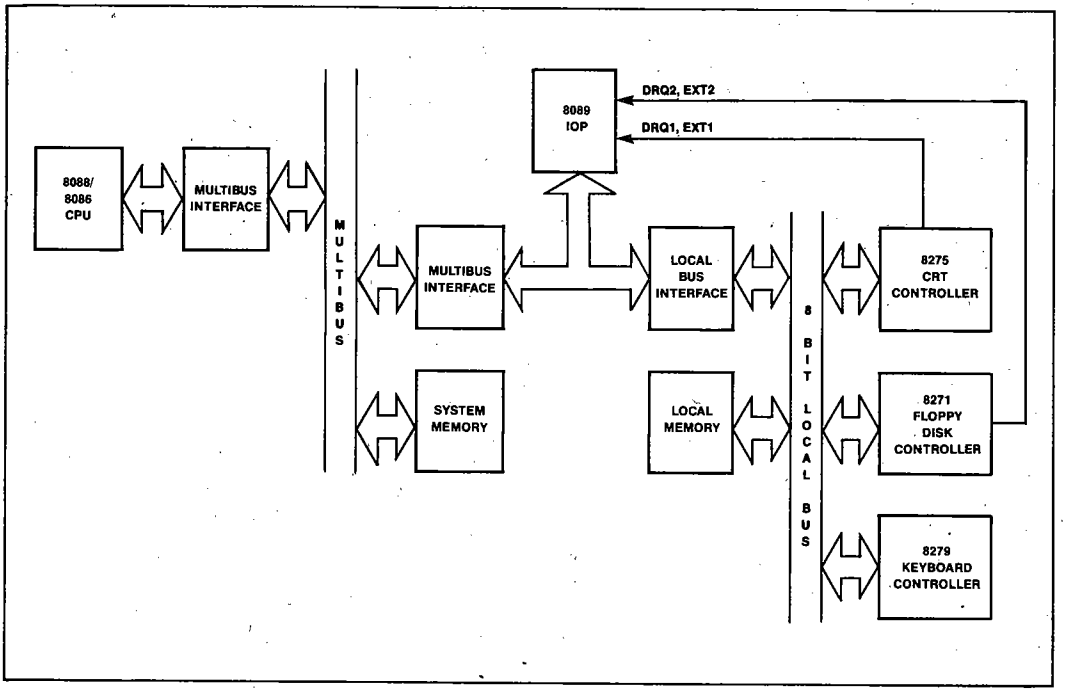


Figure 7. I/O System with Multiple Peripherals

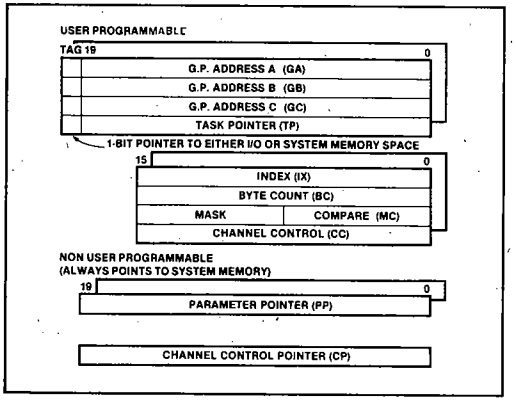


Figure 8. 8089 Register Set

System Configurations

The hardware relationship between the host CPU and the 8089 can take one of two basic forms—local configuration or remote configuration. In local configuration (Figure 9) the IOP shares the system bus interface logic with the host CPU. They reside on the same bus, sharing the same system address buffers, data buffers, and bus timing and control logic. The 8089 requests the use of the bus by activating the request/grant line to the host CPU. When the host relinquishes the bus, the IOP uses all the same hardware, and the host CPU is restricted from accessing the bus until the 8089 returns control of the bus to the host CPU.

The local configuration is a very economical configuration in terms of hardware cost, but it does not allow concurrent processing, and thus it is not able to really take advantage of the 8089's capabilities for independent operation. In the local configuration, the 8089 acts as a local DMA controller for the CPU, providing enhanced DMA capabilities and 1-megabyte addressing.

For applications such as the color graphics terminal, where system bus utilization (and other overhead) due to I/O processing would clearly be excessive in the local configuration, it is far more desirable to use the remote configuration, illustrated in Figure 10. The two processors both access the system bus, but each may have its own local bus in addition. Each of the processors may execute programs from memory on its own local bus, or

Table 1. Channel Register Summary

Register	Size	Program Access	System or I/O Pointer	Use by Channel Programs	Use in DMA Transfers
GA	20	Update	Either	General, base	Source/destination pointer
GB	20	Update	Either	General, base	Source/destination pointer
GC	20	Update	Either	General, base	Translate table pointer
TP	20	Update	Either	Procedure return, instruction pointer	Adjusted to reflect cause of termination
PP	20	Reference	System	Base	N/A
IX	16	Update	N/A	General, auto-increment	N/A
BC	16	Update	N/A	General	Byte counter
MC	16	Update	N/A	General, masked compare	Masked compare
CC	16	Update	N/A	Restricted use recommended	Defines transfer options

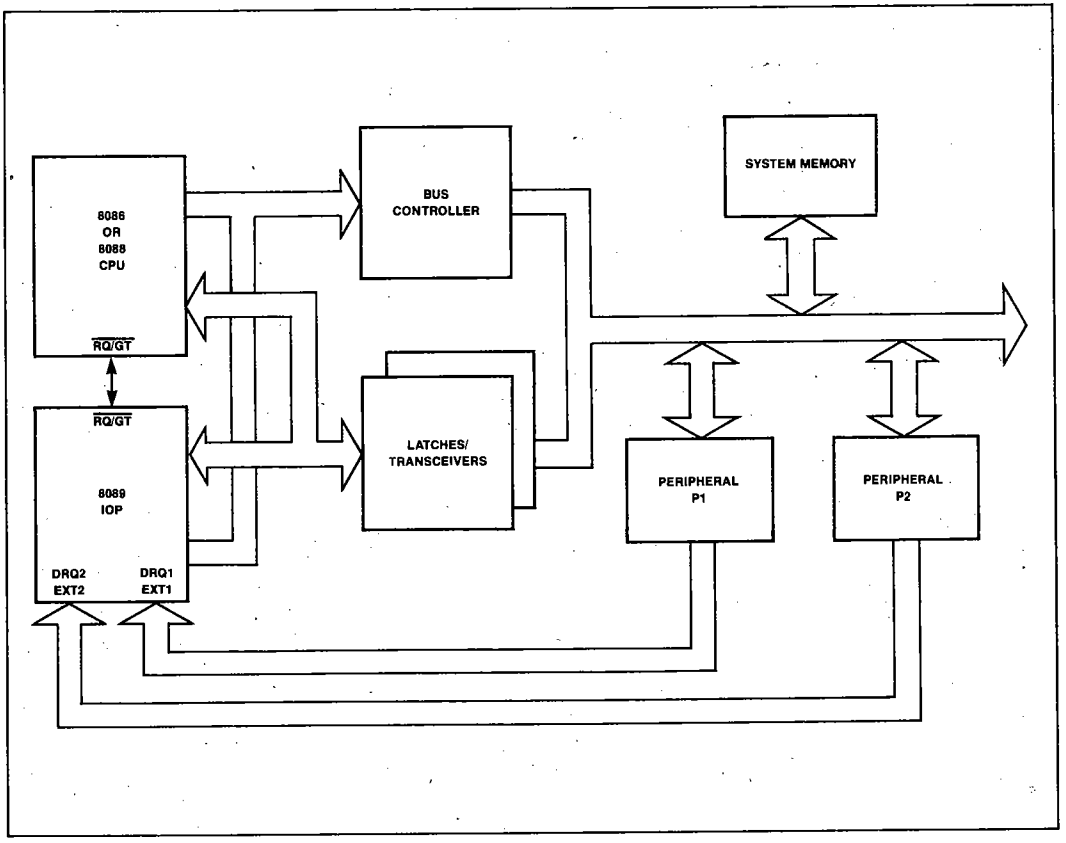


Figure 9. CPU and IOP in Local Configuration

210355-001

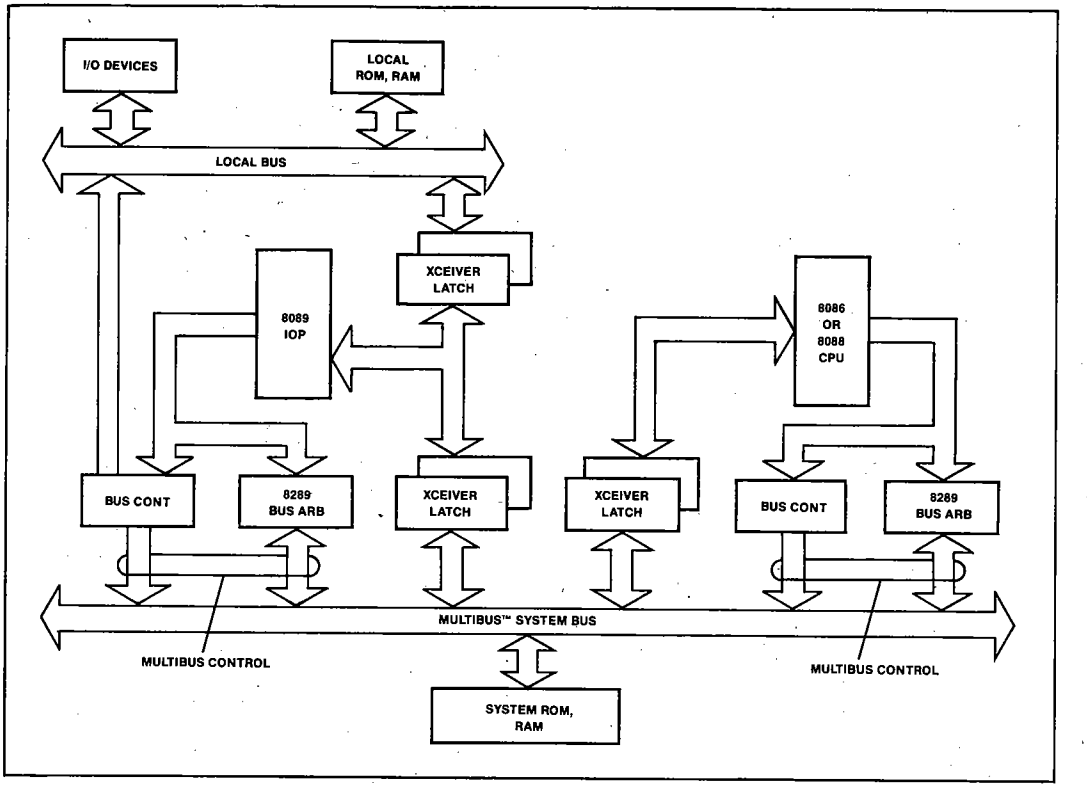


Figure 10. CPU and IOP in Remote Configuration

on the shared system bus. This creates a much more flexible arrangement. Concurrent processing may be used, and it is not necessary to synchronize the processors. An 8086, for example, may run at 8 or 10 MHz while the 8089 operates at 5 MHz. The specific terminal design described later in this application note makes use of one additional technique to further decouple the operation of the two processors. This is a dual-port RAM, which is located between the system bus and the 8089, and serves as display memory and as storage for the task blocks created by the 8086 CPU. Details on how this dual-port RAM operates are given below in the sections describing the terminal design itself.

Software Interface

Although the 8089 is an intelligent device which has a great deal of ability to function independently when managing the course of I/O operations, it typically operates under the overall supervision of the host CPU. Figure 11 illustrates the method of communication between the CPU and the IOP. The CPU communicates to the IOP by placing messages in memory and activating the IOP's channel attention (CA) input. The IOP communicates to the CPU by placing messages in system memory and making an interrupt request on one of its system interrupt request (SINTR-1 or SINTR-2) outputs.

The messages in memory take the form of linked blocks. These blocks are of the following five types:

1. System Configuration Pointer (SCP)
2. System Configuration Block (SCB)
3. Channel Control Block (CCB)
4. Parameter Block (PB)
5. Task Block (TB)

The SCP and SCB blocks are used by the CPU (only after reset) to initialize the 8089. The CCB, PB and TB blocks are used when the CPU wishes to instruct the

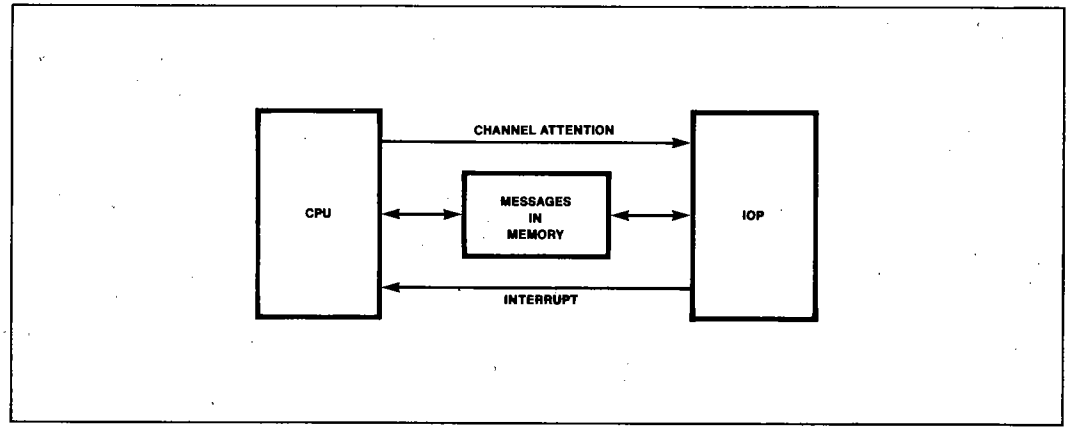


Figure 11. CPU/IOP Communication

IOP to perform a particular sequence of operations. Figure 12 shows these five blocks and how they are linked. The SCP, SCB, CB, and PB must be in memory which is accessible from both the CPU and 8089 (either system memory or for this application note, dual-port memory). The TB may be in either system or 8089 local memory.

The system configuration pointer is always found at the same location (FFFF6) in the system memory. The first time channel attention is activated (after an IOP reset) the 8089 reads the system configuration pointer from this location. The SYSBUS field contains only one significant bit (Bit 0), designated by the letter W. If W = 0, the system bus is an 8-bit bus. W = 1 denotes a 16-bit system bus. The IOP first assumes an 8-bit bus and reads the SYSBUS field. It stores the information as to the physical width of the system bus, then immediately uses this information in the process of fetching the next four bytes, which contain the address of the system configuration block.

The addresses used to link blocks are standard iAPX 86, 88 pointer variables, each occupying two word locations in system memory. The lower-addressed word contains an offset, which is added to the segment base value (left-shifted four places) found in the upper-addressed word to derive the complete 20-bit physical address in system memory. If the block is in an I/O memory (as a task block might be), only the offset value is used.

After thus deriving the address of the system configuration block, the IOP reads this block, starting with the system operation command (SOC) field. Bit 1 of the SOC field specifies the request/grant mode (used in local configuration or in multiple-IOP systems). Bit 0 specifies the I/O bus width (designated I). When I = 0, the I/O bus is an 8-bit bus. I = 1 denotes a 16-bit I/O bus. The IOP then proceeds to read the double-word pointer to the channel control block, converts it to the 20-bit physical address, and stores it in an internal register (the channel control pointer register). This register is loaded only during initialization and is not available to channel programs. For this reason the channel control block cannot be moved unless the IOP is reset and reinitialized.

The initialization is complete when the channel control pointer has been stored. The IOP indicates this by clearing the busy flag in the channel 1 control block (which must be set by the host CPU before the initialization sequence began). The host CPU can monitor this flag to determine when initialization is complete, and then to initialize any other 8089s in the system.

It is the responsibility of the host CPU to make sure that the SCP and SCB have the proper contents before issuing the channel attention (CA) that begins the initialization sequence. After initialization, the host CPU must also assure that the channel control block (CCB), parameter block (PB), and task block (TB) all have the proper contents, before issuing a subsequent CA.

The CA may be issued in the form of an I/O write command to the address of the IOP on the Multibus. Figure 13 shows a typical decoding circuit for this write command. The IOP actually occupies two consecutive address locations on this bus, because the A0 line is tied to the select (SEL) input of the 8089. A zero on the SEL line specifies IOP channel 1 for the impending operation, while a one specifies IOP channel 2.

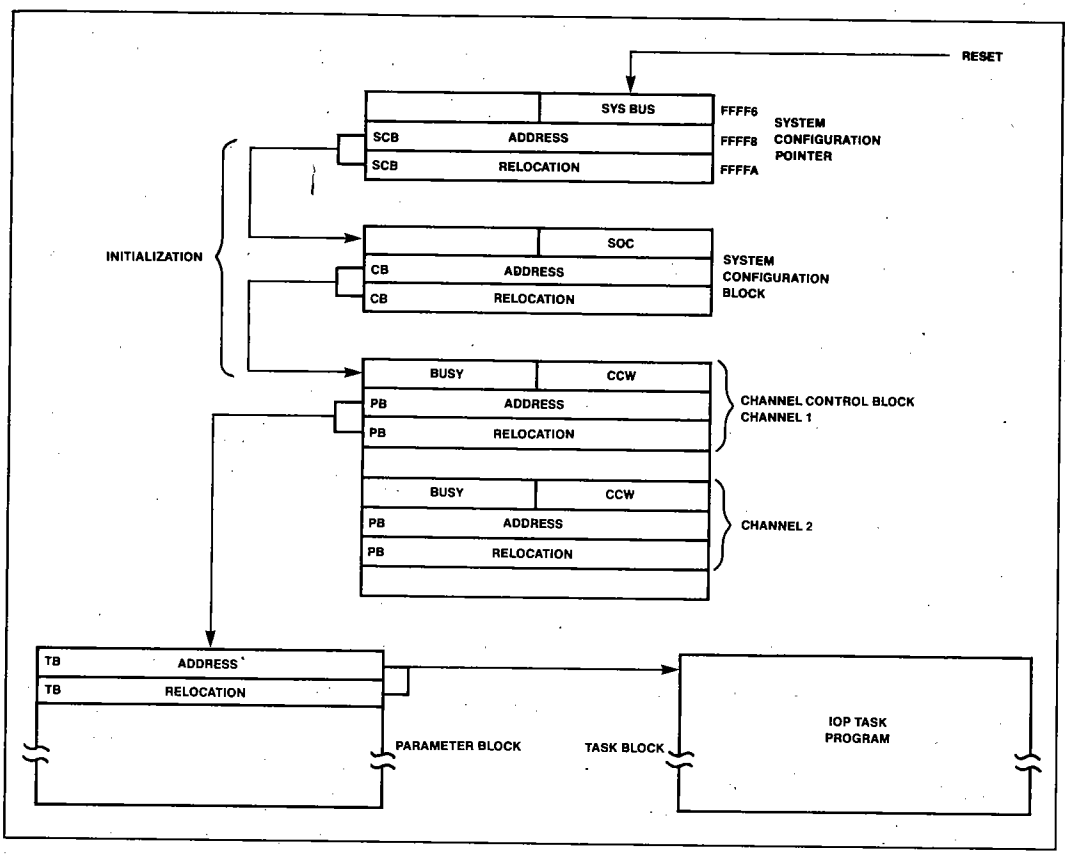


Figure 12. Linked Block Communication Structure

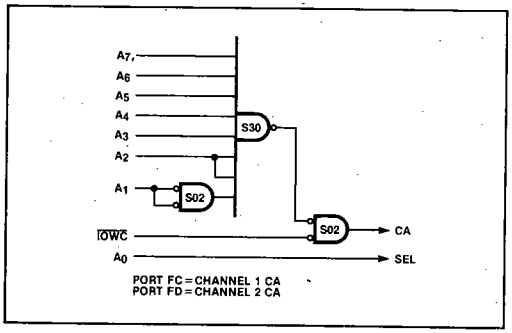


Figure 13. Channel Attention Decoding Circuit

The channel control block has a section for each channel. When the CA is received, the IOP goes to the section corresponding to the selected channel, and reads the channel command word (CCW). It then sets or clears the busy flag (FFH = set, 00H = clear). The encoding of the channel command word is shown in Figure 14. The CCW provides the IOP with a functional command (START in I/O space, HALT, etc.) and specifies some of the operating conditions, such as interrupt handling, bus load limit, or priority relative to the other channel. If the CPU is instructing the IOP to execute a program, it is at this point that the CPU specifies, via the CCW, whether the instructions are to be fetched from the system space or from the 8089's I/O space. Refer to *iAPX 86,88 User's Manual* for specific details on the setting and clearing of the busy flag and on CCW specifications.

After the CCW has been read, the IOP reads (if appropriate to the command) the address of the parameter block associated with the impending operation, and stores the translated address (from the two-word segment and offset pair to the 20-bit physical address) in

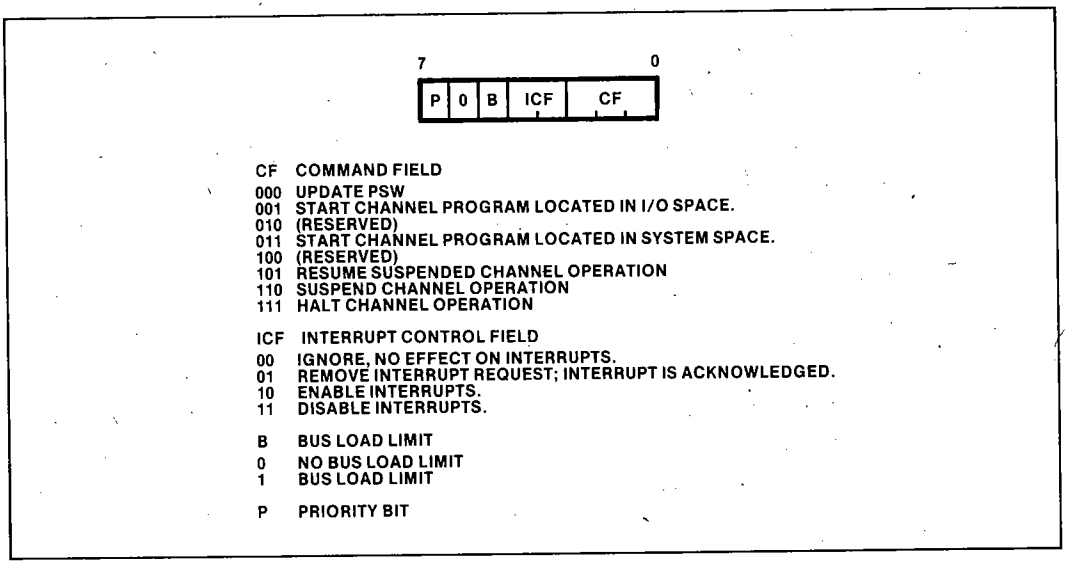


Figure 14. Channel Command Word Encoding

the parameter pointer (PP) register. PP is another register which is not programmable by the channel program. The IOP then goes to this location in system memory, and fetches the address of the task block itself. The task block contains the actual program to be executed, while the parameter block contains parameters to be used by that program.

Except for the first two words, which contain the task block address, the parameter block format is up to the discretion of the user. Similarly, the task block may have any format whatsoever, as long as the IOP can execute the program. The parameter block is always in system memory, but the task block may be either in system memory or in I/O (local) memory.

The host CPU may prepare as many parameter-block/task-block sets as it wishes. An individual set is then activated for execution by placing its parameter block pointer in the desired channel's control block, loading the appropriate channel control word, and issuing a CA to that channel.

The registers shown in Figure 8 store (in addition to pointer variables) various flags and parameters associated with the IOP's operation. Some of these registers are loaded automatically with information fetched during the initialization sequence or during channel attention processing. Others must be set by executing a program using instructions from the IOP's instruction set that are specifically designed for loading these registers.

Channel programs (task blocks) are written in ASM-89, the 8089 assembly language. About 50 basic instructions are available. The IOP instruction set contains some instructions similar to those found in CPUs, and also other instructions specifically tailored to I/O operations. Data transfer, simple arithmetic, logical, and address manipulation operations are available. Unconditional jump and call instructions are provided so that channel programs can link to each other. An individual register or even a single bit may be set or cleared with a single instruction. Other instructions specify conditional jumps, initiate DMA transfers, perform semaphore operations, and issue interrupt requests to the CPU.

A channel program typically ends by posting the result of an operation to a field supplied in the parameter block, then interrupting the CPU (if interrupts are enabled) and halting. When the channel halts, its associated BUSY flag is cleared in the channel control block. The CPU can poll this flag (as an alternative to being interrupted) to determine when the operation has been completed.

Timing Details

The basic bus timing relationships for the 8089 are identical to those of the 8086 or 8088, in that all cycles consist of four states (assuming no wait states), and use the same time-multiplexing technique for the address/data lines. The address (and ALE signal from the

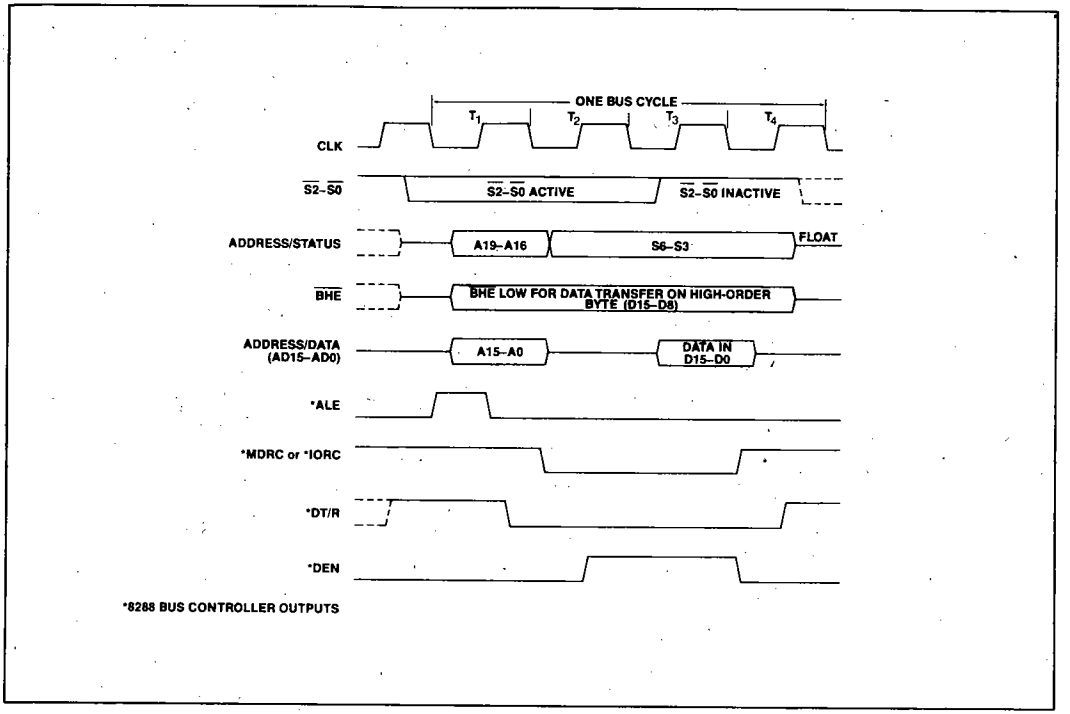


Figure 15. Read Bus Cycle

8288 bus controller) is output during state T1 for either a read or write cycle. During state T2 for a read cycle (Figure 15) the address/data lines are floated. During state T2 for a write cycle (Figure 16) data is output on these lines. During state T3, the write data is maintained or the read data is sampled. The bus cycle is concluded in state T4.

Figure 17 shows some details on the wait state timing and Figure 18 shows the RESET-CA initialization timing.

During DMA transfers, the transfer cycle may be synchronized by either the source or the destination. Figure 19 (source-synchronized transfers) and Figure 20 (destination-synchronized transfers) show the relationships among the basic clock cycles, the DRQ signals, and the DACK signals.

The 8089 does not have a DACK output signal. Rather, it uses the more general process of issuing a command (for example, I/O read or write) to an address on the I/O bus. This command is then hardware decoded to obtain a chip select signal for the addressed device. This method enables the 8089 to relate to a variety of I/O devices in a very flexible manner.

Figures 19 and 20 also show how the 8089 inserts idle clocks to accommodate various DRQ latency conditions. If maximum efficiency (transfer rate) is desired, it is usually possible to remove this latency by techniques such as generating an early DRQ. Another possibility is to use the unsynchronized DMA transfer mode (DRQ is not examined) and to use the READY signal for synchronizing transfers. The early DRQ technique will be discussed later.

GRAPHIC CRT SYSTEM DESIGN

Having examined the requirements for graphic CRT systems in general, and having also discussed the capabilities of the 8089, we can now proceed to describe a specific graphic CRT design using the 8089.

In this design, the system CPU is an 8086. Thus, the entire system is called an Intel 8089.

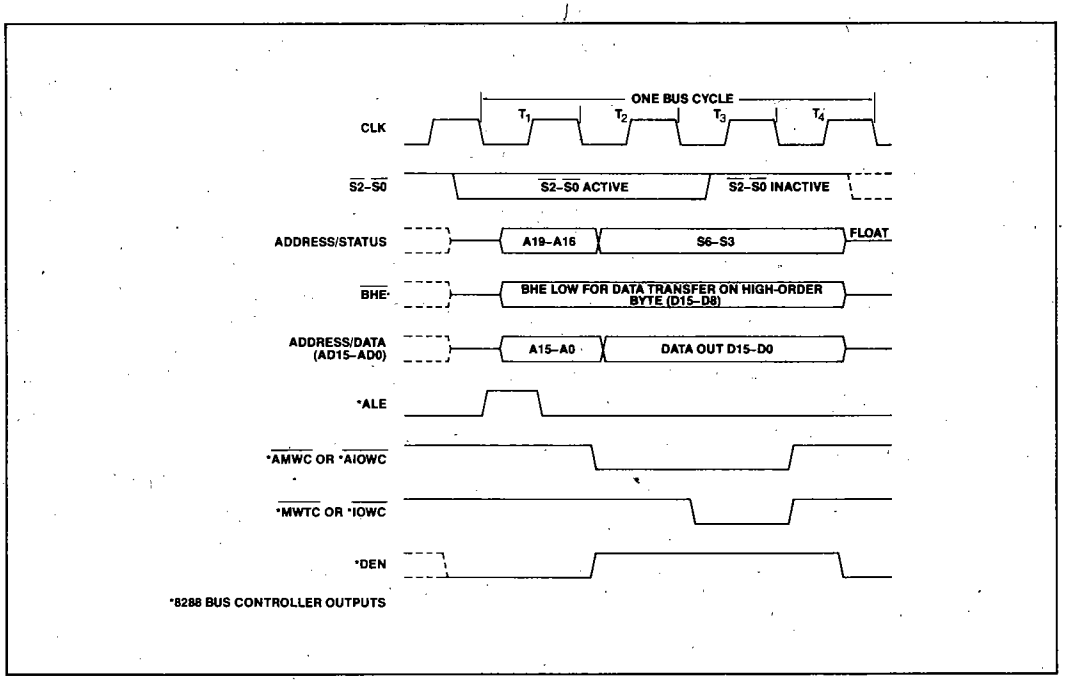


Figure 16. Write Bus Cycle

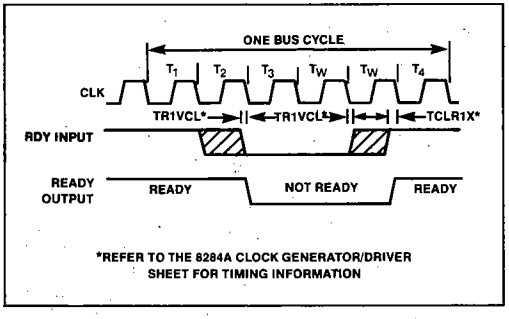


Figure 17. Wait-State Timing (Synchronous RDY Input)

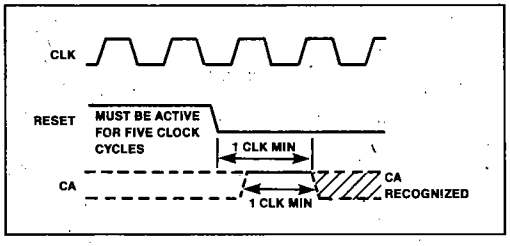


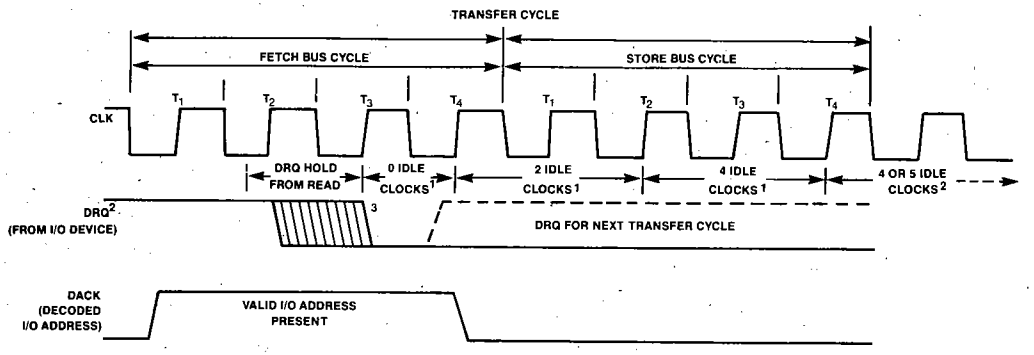
Figure 18. Reset and Channel Attention Timing

System Partitioning

The 8086 and 8089 are arranged in the remote configuration. This assures that concurrent processing can occur. As mentioned earlier, an additional step is taken to further decrease system bus utilization for I/O-related processes. This step is the inclusion in the system of a dual-port RAM, located between the system bus and the 8089. This dual-port RAM contains the display memory and also contains the linked message blocks used for communication between the 8086 and the 8089.

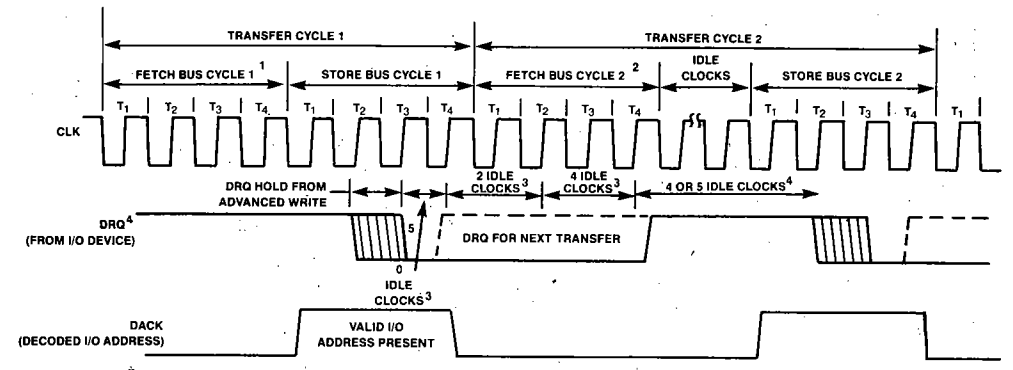
The system configuration then becomes that shown in Figure 21. The dual-port RAM becomes the only data path between the 8086 and the 8089. Access to this memory is time-shared between the 8086 and the 8089, with the 8089 taking less than 50% of the total time available. Since the 8089 does not access the system bus, the host system can enjoy complete freedom to allocate its resources between its own local bus and the system bus. The CPU and the IOP can operate asynchronously, with the 8086 running on an 8 MHz clock and the 8089 on a 5 MHz clock.

The division of responsibility between the 8086 and the 8089 is then very clearly defined. The 8086 initializes the 8089 and specifies the task parameters, storing them in



NOTES:
1) INDICATES THE NUMBER OF IDLE CLOCKS INSERTED AFTER T_4 OF THE STORE CYCLE BEFORE THE NEXT TRANSFER CYCLE BEGINS. IF DRQ IS RECEIVED BEFORE THE RISING EDGE OF CLK IN THE CURRENT FETCH CYCLE, THE NEXT FETCH BEGINS IMMEDIATELY AFTER THE CURRENT STORE.
2) IF THE 8089 IS IDLE WHEN DRQ IS RECOGNIZED, FOUR OR FIVE MORE IDLE CLOCK CYCLES OCCUR BEFORE THE ASSOCIATED TRANSFER CYCLE BEGINS. (DRQ IS LATCHED ON THE RISING EDGE OF CLK.)
3) TO PREVENT THE START OF THE NEXT TRANSFER CYCLE, DRQ MUST BE BROUGHT LOW BY THE RISING EDGE OF CLK IN T_4 OF THE CURRENT FETCH. (FOR B/B→W SOURCE SYNCHRONIZED AND W→B/B DESTINATION SYNCHRONIZED IT MUST BE LOW BY THE RISING EDGE OF CLK IN THE FOURTH CLOCK OF THE CURRENT BUS CYCLE INCLUDING WAIT STATES.)

Figure 19. Source-Synchronized Transfer Cycle



NOTES:
1) FIRST DMA FETCH CYCLE OCCURS IMMEDIATELY AFTER THE LAST TASK PROGRAM INSTRUCTION IS EXECUTED.
2) FETCH CYCLE 2 BEGINS IMMEDIATELY AFTER STORE CYCLE 1.
3) INDICATES THE NUMBER OF IDLE CLOCKS INSERTED AFTER T_4 OF THE FETCH BEFORE STORE CYCLE 2 BEGINS. IF DRQ IS RECEIVED BEFORE THE RISING EDGE OF CLK IN THE CURRENT STORE CYCLE, THE NEXT STORE BEGINS IMMEDIATELY AFTER THE NEXT FETCH.
4) IF THE 8089 IS IDLE WHEN DRQ IS RECOGNIZED, FOUR OR FIVE MORE IDLE CLOCK CYCLES OCCUR BEFORE THE ASSOCIATED STORE CYCLE BEGINS. (DRQ IS LATCHED ON THE RISING EDGE OF CLK.)
5) TO PREVENT THE NEXT STORE CYCLE FROM OCCURRING, DRQ MUST BE BROUGHT LOW BY THE RISING EDGE OF CLK IN T_4 OF THE CURRENT STORE (FOR B/B→W SOURCE SYNCHRONIZED AND W→B/B DESTINATION SYNCHRONIZED, IT MUST BE LOW BY THE RISING EDGE OF CLK IN THE FOURTH CLOCK OF THE CURRENT STORE CYCLE INCLUDING WAIT STATES.)

Figure 20. Destination-Synchronized Transfer Cycle

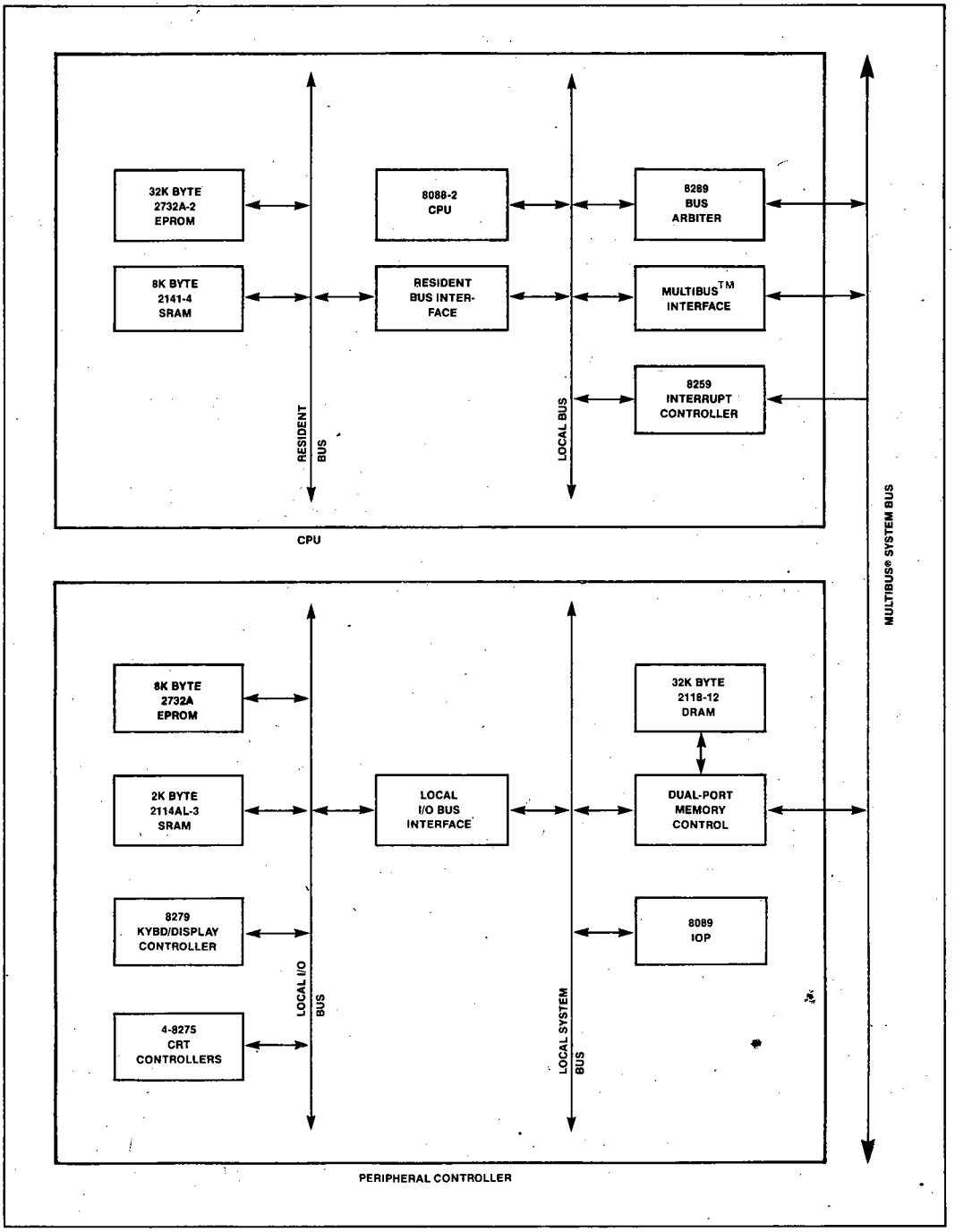


Figure 21. Remote Configuration with Dual-Port RAM

210355-001

the dual-port RAM. In many cases, the 8086 also prepares the task programs and stores them in the dual-port RAM, from which they may be downloaded to a memory on the 8089's I/O bus. The 8089 executes the task programs (from the dual-port RAM or from a local memory on the I/O bus), while the 8086 simultaneously executes other control or application programs. The application programs may encompass a wide variety of operations, but they will always generate the display characters and store them in the dual-port RAM. The 8089 returns status to the 8086 when task program execution has been completed.

Figures 22 and 23 show the manner in which the memories are organized. Figure 22, which shows the memory configuration for the 8086, should be taken as an example, since many different configurations are possible, according to the user's application. Figure 23 shows the memory configuration for the 8089, given the particular choices made for the application discussed in this note. Of the memories shown in Figure 22, the 2141 static RAMs and the 2732A EPROMs are located on the 8086's local bus, while the 2816 EEPROM and the 2118 dual-port RAM are interfaced to the Multibus. The 2816 is a non-volatile read/write memory equivalent in its storage capacity to the 2716 EPROM.

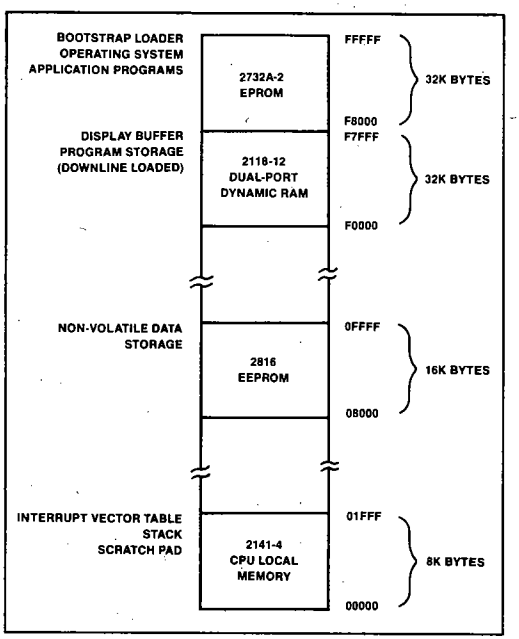


Figure 22. CPU Memory Organization

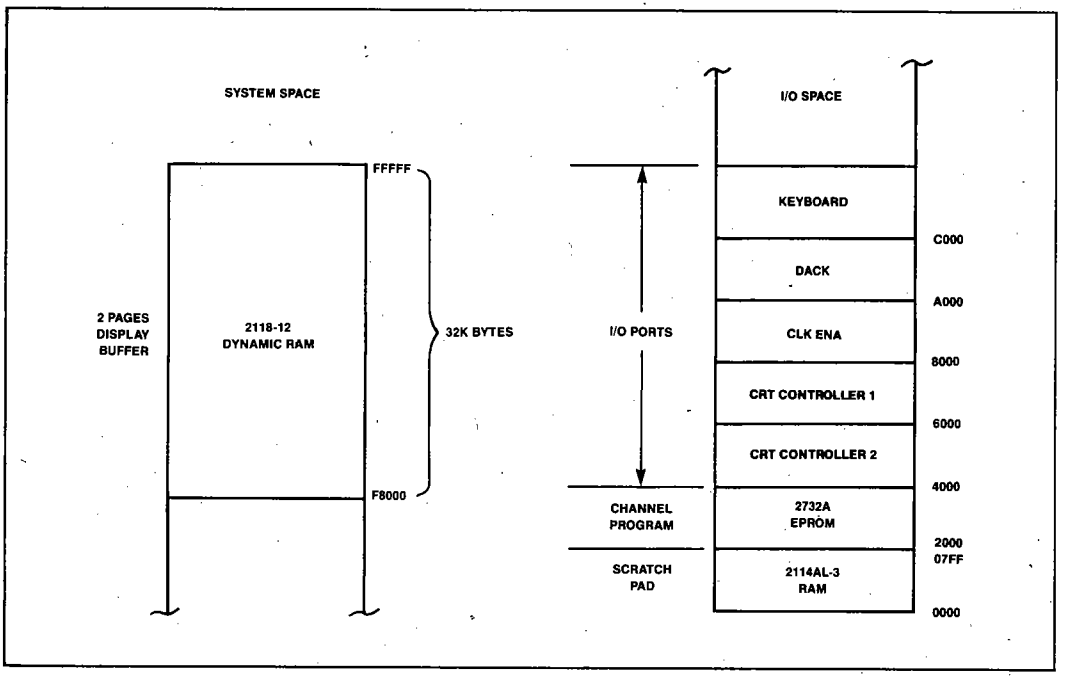


Figure 23. IOP Memory Organization

8086/8089 Software Interface

Comparing Figures 22 and 23, it can be seen that the 2118 dynamic RAM appears in the memory configurations for both the 8086 and the 8089. In the 8086's system space, this memory occupies addresses F0000 through F7FFF, while in the 8089's system space, its address range is F8000 through FFFFF.

Figure 24 shows the organization of the dual-port RAM. The addresses given are those seen by the 8089. The display data (for the CRT refresh function) is contained in the two largest blocks—Display Page 0 and Display Page 1. Each page contains 15K bytes, enough to refresh a color graphic screen containing 48 rows of 80 characters each. In typical operation, the 8086 and the 8089 both access the same page of display data. In special cases, such as animated displays, the 8089 performs repetitive DMA transfers from one of these pages, while the 8086 is generating new display material and storing it in the other page. The display page pointer (DSPLY_PG_PTR) in the parameter block specifies which of these pages is to be displayed at any given time. This pointer may be changed by the 8086, or by a command from the terminal keyboard.

The Command Buffer is a 256-byte area set aside for transferring ASCII characters from the 8086 to the 8089. It is like a second keyboard, scanned by the 8089. It takes precedence over any real keyboard activity. The COM_8086 flag in the parameter block is used to indicate when there are entries in the command block area.

The EEPROM Buffer is a 256-byte area used in connection with the non-volatile EEPROM memory, an optional memory which may be located on the Multibus. One use of such a memory would be to store ASCII strings, which could then be recalled by the 8086 upon recognition of special keyboard control code sequences.

The Keyboard Buffer is a 256-byte area which serves as a storage area for ASCII characters entered from the terminal keyboard. When this buffer becomes full, or when a return is entered at the keyboard, an end-of-file byte is placed after the last entered character, and the keyboard buffer full (KBD_BUF_FULL) flag is set in the parameter block. This prevents the 8089 from processing any more inputs from the keyboard, until the 8086 resets KBD_BUF_FULL.

The Spare blocks total 1K (1024) bytes, and may be used for any purpose, according to the user's application.

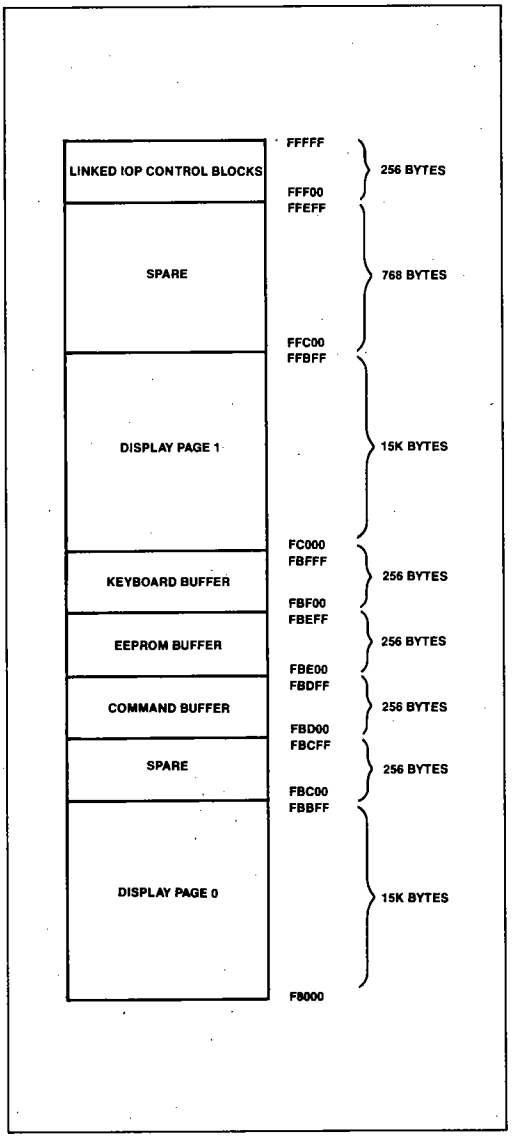


Figure 24. Organization of the Dual-Port RAM

The Linked IOP Control Blocks are those which have been discussed above, as part of the 8089 overview. The specific memory locations are as shown in Figure 25. Note that there is only one parameter block, and no task blocks present. Only one task block is used in this application, and it is stored in the 2732A EPROMs on the 8089's I/O bus.

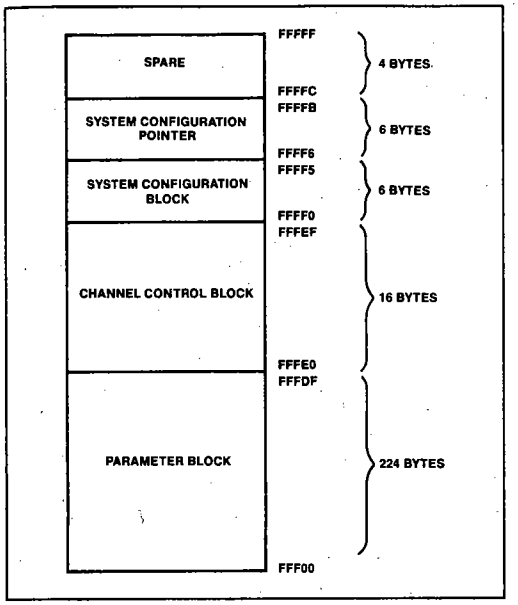


Figure 25. Organization of the Linked IOP Control Blocks Area

As mentioned earlier, the structure of the parameter block is very flexible. Only the first four bytes are fixed (because of the 8089's requirements). These four bytes contain the address of the task block. The remaining space in the parameter block may be defined by the user. The following list shows the parameter block structure that is used in support of the channel program contained in the 2732A EPROMs on the 8089's I/O bus.

TP_LSW	DW
TP_MSD	DW
EEP_INH	DB
EEP_BUF_FULL	DB
EEP_RECALL	DB
COL_CH_INH	DB
KBD_INH	DB
KBD_BUF_FULL	DB
COM_8086	DB
COLOR	DB
STR_PTR_8086	DW
BACK_COL_SW	DB
MON_INH	DB
DSPLY_PG_PTR	DB
SCROLL_REQ	DB
MON_HOM	DW
MON_END	DW
MON_LMARG	DW
MON_RMARG	DW
KBD_BUF_PTR	DW

In the above table, DB represents a one-byte quantity, and DW represents a two-byte quantity.

TP_LSW and TP_MSD are the two words making up the task pointer. However, since in this application the task program is in the I/O space, only the least-significant word (LSW) is fetched.

EEP_INH, when not equal to zero, indicates that the EEPROM buffer is closed to keystrokes or 8086 ASCII commands.

EEP_BUF_FULL, when not equal to zero, indicates that the EEPROM buffer is full.

EEP_RECALL, when not equal to zero, indicates that the 8089 is recalling the contents of an EEPROM buffer area.

COL_CH_INH, when not equal to zero, inhibits the color control keys on the keyboard.

KBD_INH, when not equal to zero, inhibits the processing of keystrokes (entered at the keyboard) by the 8089. Up to 6 keystrokes may be saved in the keyboard controller and may be processed later.

KBD_BUF_FULL, when not equal to zero, indicates that a new line of keyboard data needs to be processed by the 8086. The 8089 sets KBD_BUF_FULL equal to −1 when the return key is pressed. The 8086 resets KBD_BUF_FULL to zero after it has read this data.

COM_8086, when not equal to zero, indicates that there are ASCII commands in the command buffer areas of dual-port RAM that need to be processed by the 8089.

COLOR determines the foreground and background colors to be used in connection with ASCII characters entered at the keyboard, or sent by the 8086 via the command buffer area. In the COLOR byte, bits B0–B2 determine the background color, while B3–B5 determine the foreground color. The following code is used:

000	Black
001	Red
010	Green
011	Yellow
100	Blue
101	Magenta
110	Cyan
111	White

STR_PTR_8086 is a two-byte quantity that serves as an offset address for the ASCII characters in the command buffer.

BACK_COL_SW determines whether the 8089 color control keys will alter the foreground or the background portions of the COLOR byte. If BACK_COL_SW equals zero, the foreground color is altered. If BACK_COL_SW is not equal to zero, the background color is altered.

MON_INH, when not equal to zero, suspends DMA transfers by the 8089 from display memory to the 8275s. When MON_INH is cleared, DMA will resume.

DSPLY_PG_PTR determines which of the two display pages will be used to refresh the CRT. If DSPLY_PG_PTR equals zero, page 0 will be displayed. If DSPLY_PG_PTR does not equal zero, page 1 will be displayed.

SCROLL_REQ is set by the 8089 to indicate to the 8086 that the cursor is at the bottom of the page, and that key entry/command processing has been halted, pending a display memory scroll operation. When the 8086 has performed this operation, it clears SCROLL_REQ.

MON_HOM, MON_END, MON_LMARG, and MON_RMARG specify, respectively, the upper, lower, left, and right boundaries of the region on the screen in which keyboard entries will be displayed.

KBD_BUF_PTR is a two-byte quantity that serves as an address for the ASCII characters in the keyboard buffer.

Note that a number of these parameters support options (e.g., EEPROM buffer) and are not critical to the graphic operation described in this application note.

8089 Display Hardware Interface

This section describes the hardware of the peripheral processing module (PPM), which includes everything between the system bus and the CRT display/keyboard unit. The overall organization of the PPM is as shown in Figure 21. The dual-port RAM can be accessed from either the system bus or the 8089's local bus. The 8089 is said to be operating in the system space when it is accessing the dual-port RAM, and in the I/O space when it is accessing devices on the I/O bus. Included on the I/O bus are four 8275 CRT controllers, an 8279-5 keyboard controller, two 2732A EPROMs, which are used to hold channel programs, and four 2114 static RAMs, which are used as scratch-pad RAM for the 8089.

As explained above (under *OVERVIEW OF CRT GRAPHIC SYSTEMS, Performance Requirements)*, four bytes are used to specify each character in the display. The first byte determines whether the character is a text character or a graphic character, and specifies the colors for foreground and background. If it is a text character, the second byte specifies the character with a seven-bit ASCII code, and the remaining two bytes are not used. If it is a graphics character, the second, third, and fourth bytes contain the color specification for each of the twenty distinct picture elements (pixels) within the character. Use of the foreground color is indicated by a one in the respective bit position, while a zero specifies use of the background color.

The structure of the display characters and the formats of the individual bytes are shown in Figures 4 and 5.

The four 8275 CRT controllers on the 8089's I/O bus are used to process the four bytes comprising each character. Since the 8089 can transfer two bytes at a time in DMA mode, the four bytes are transferred in two stages. In the first stage, the 8089 fetches the first two bytes from the dual-port RAM, and transfers these two bytes into the first pair of CRT controllers. In the second stage, the 8089 fetches the second two bytes from the dual-port RAM, and transfers these two bytes into the second pair of CRT controllers. This process is repeated 80 times to transfer the 80 characters making up each row in the display.

The distinction between text and graphic characters is entirely transparent to the 8089. Four bytes are transferred in every case, even though the text information only requires two bytes per character.

We shall now examine the hardware schematics in detail, to see how the various functions of the PPM are implemented. Figure 26 shows the 8089 IOP and its associated bus controller. At the top left are the inputs through which the 8089 is controlled. The DRQF signal (detailed later) is the DMA request that initiates the transfer of two bytes from the IOP to two of the four CRT controllers. DRQF comes from the 8275s via a one-shot, and is connected to the DRQ 1 input of the 8089.

IRQ is an interrupt request that comes from the 8275s. It is activated after an entire screen's video information has been transferred from the dual-port RAM to the 8275s. IRQ is connected to the EXT 1 input of the 8089. It is necessary to program the 8089 to terminate the DMA transfer on an external event, in order for this signal to be effective.

CA is the channel attention signal. Upon receipt of CA, the 8089 reads the channel control word (CCW) from the dual-port RAM. From the CCW, the 8089 determines the nature of the operation assigned to it by the

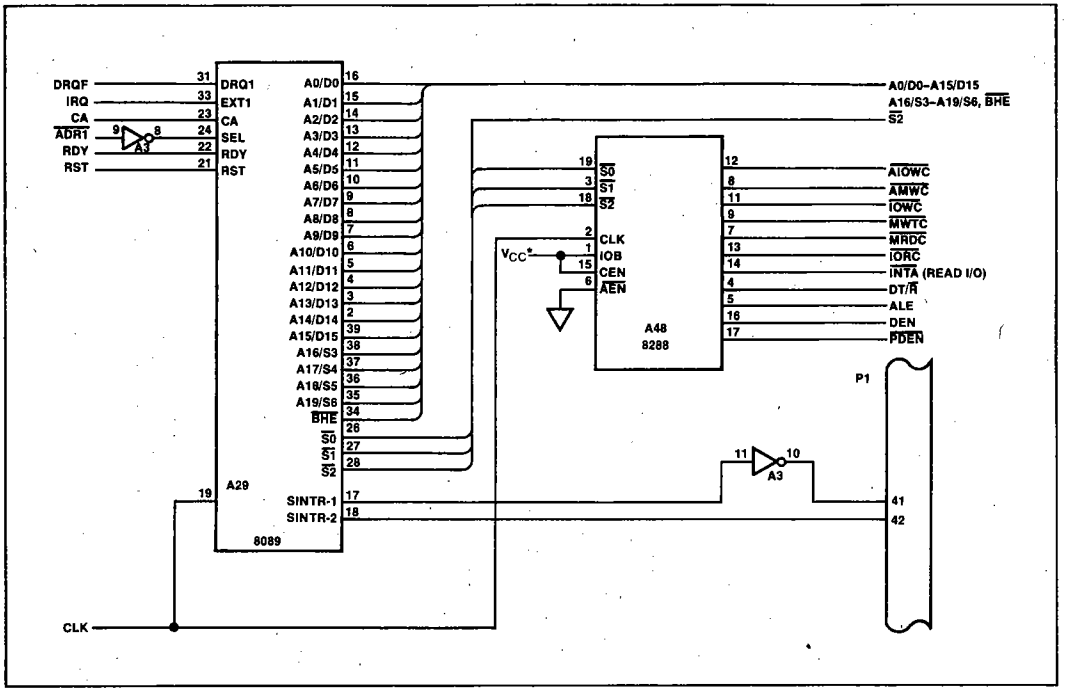


Figure 26. 8089 I/O Processor and 8288 Bus Controller

8086. CA is derived by hardware decoding of an I/O write command made by the 8086 to address 00H or address 01H on the Multibus. The lowest-order bit of this address is used to specify whether channel 1 or channel 2 of the IOP is to be selected, and is connected to the 8089's SEL input. In this application, the DMA transfers are always performed by channel 1.

RDY is the ready signal that comes from the 8202 dynamic RAM controller, and is synchronized by the 8284A clock generator. RDY is low whenever the 8086 is accessing the dual-port RAM. The RDY signal is used to establish a master/slave relationship between the 8086 and the 8089, with the 8086 as the master. As mentioned earlier, the 8089 accesses the dual-port RAM about 50% of the time during DMA transfers. It can be seen, referring to Figure 20, that if no idle clocks occur, the IOP will access the dual-port RAM during the four clock times of the DMA-fetch bus cycle, and will access the I/O bus during the four clock times of the DMA-store bus cycle. While the 8089 is doing the store operation, the 8086 can access the dual-port RAM. Once the 8086 has gained this access, the RDY signal will remain low until the 8086 is finished. The 8089 waits for RDY to go high before making a subsequent fetch.

At 5 MHz, the 8089 requires 3.2 microseconds (16 clock cycles) to transfer the four bytes representing a graphic character from the display memory to the four 8275s, assuming that no wait states have been inserted because of the 8086's access to the dual-port RAM, or because of dynamic RAM refresh functions. A complete row, consisting of 80 characters, requires 80 x 3.2 = 256 microseconds. The time allowed to complete the transfer of one row must be less than the time it takes to display that row on the screen. This latter time is equal to 1/50 of the total screen update time, or 1/3000 of a second (333 microseconds). Comparing the two figures (256 vs 333), it can be seen that there are 77 microseconds available for such wait states. It is the responsibility of the software designer to control the 8086's access to dual-port RAM in such a mannner that the added wait states do not total more than 77 microseconds in any span of 333 microseconds. Otherwise, underruns may occur and the CRT screen will be blanked. See *System Performance* (below) for further discussion on this effect.

RST is the IOP reset signal, which comes from the 8284A clock generator. The first CA after RST causes the IOP to access address FFFF6 in the dual-port RAM, in order to read the system configuration pointer.

Outputs from the IOP are the time-multiplexed address and data lines, BHE/(bus high enable), status line S0, S1, and S2, and the system interrupt request lines, SINTR-1 and SINTR-2. The interrupt lines go directly to the MULTIBUS, and from there they become inputs to the 8086's 8259A interrupt controller.

Figure 27 shows the I/O address latches and decoder, and the circuitry used to generate the DACK/ signals for the CRT controllers. The IOP status bit S2 indicates whether the IOP is accessing the I/O space or the system space. Latched by ALE (address latch enable), S2/ generates IO and IO/. IO and IO/ are used to indicate that the 8089 is not accessing dual-port RAM. IO/ goes to the dual-port RAM controller.

The DACK/ signals are generated in the following manner:

1. Both 8275 pairs are accessed by the 8089 (DMA mode) via port A000H.
2. Hardware is used to select one pair of CRT controllers (bytes 0 and 1 or bytes 2 and 3).
3. As the 8089 reads (DMA) the word from the dual-port memory, address bit 1 (SA1) is latched with the memory read command (MRDC/).
4. When SA1 = 0, DACK 1/ is activated.
5. When SA1 = 1, DACK 2/ is activated.
6. In this manner the 8089 performs alternating writes (DMA) to the 8275 pairs.

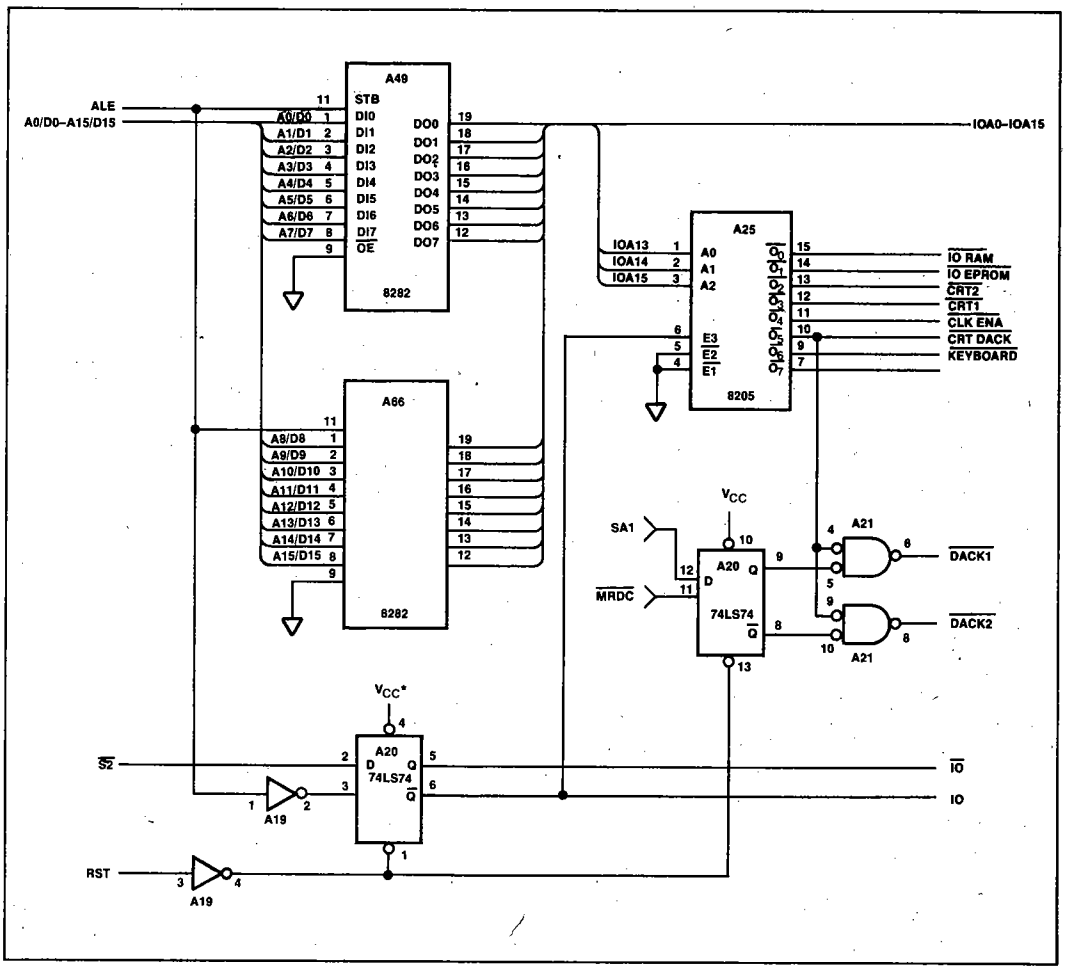


Figure 27. Address Latches, Decoders, and DACK Generator

210355-001

Figure 28 shows the bus transceivers used between the 8089 and the I/O bus, and also shows the 2732 EPROMs.

Figure 29 shows the 2K bytes of 2114 static RAM on the I/O bus, which are used as scratch-pad RAM for the 8089.

Figure 30 shows the 8279-5 keyboard controller, and also shows the 8284A clock generator that produces the CLK, RDY, and RST signals for the 8089. For more information on interfacing the 8279-5 to the keyboard (Cherry Electrical Products B70–05AB), refer to the 8279/8279-5 data sheet and application note AP-32, *CRT Terminal Design Using the Intel 8275 and 8279*.

Figure 31 shows the clock generator for the character timing and dot timing. The character clock frequency (C CLK) is 1/8 of the dot clock frequency (D CLK), 10.8 MHz. Also shown in Figure 31 is a 9602 one-shot used to generate the video sync pulses.

Figure 32 shows the CRT Controllers #0 and #1. Bit 6 of Byte 0 determines whether the display character is text or graphic. If Bit 6 is low, the character is a text character, and Byte 1 is used to address the 2732A character generator ROM. Bytes 2 and 3 are ignored. The line count outputs LC0–LC3 of an 8275 (any 8275 can be used, since they are all synchronized) are also applied to the character generator to perform the line select function.

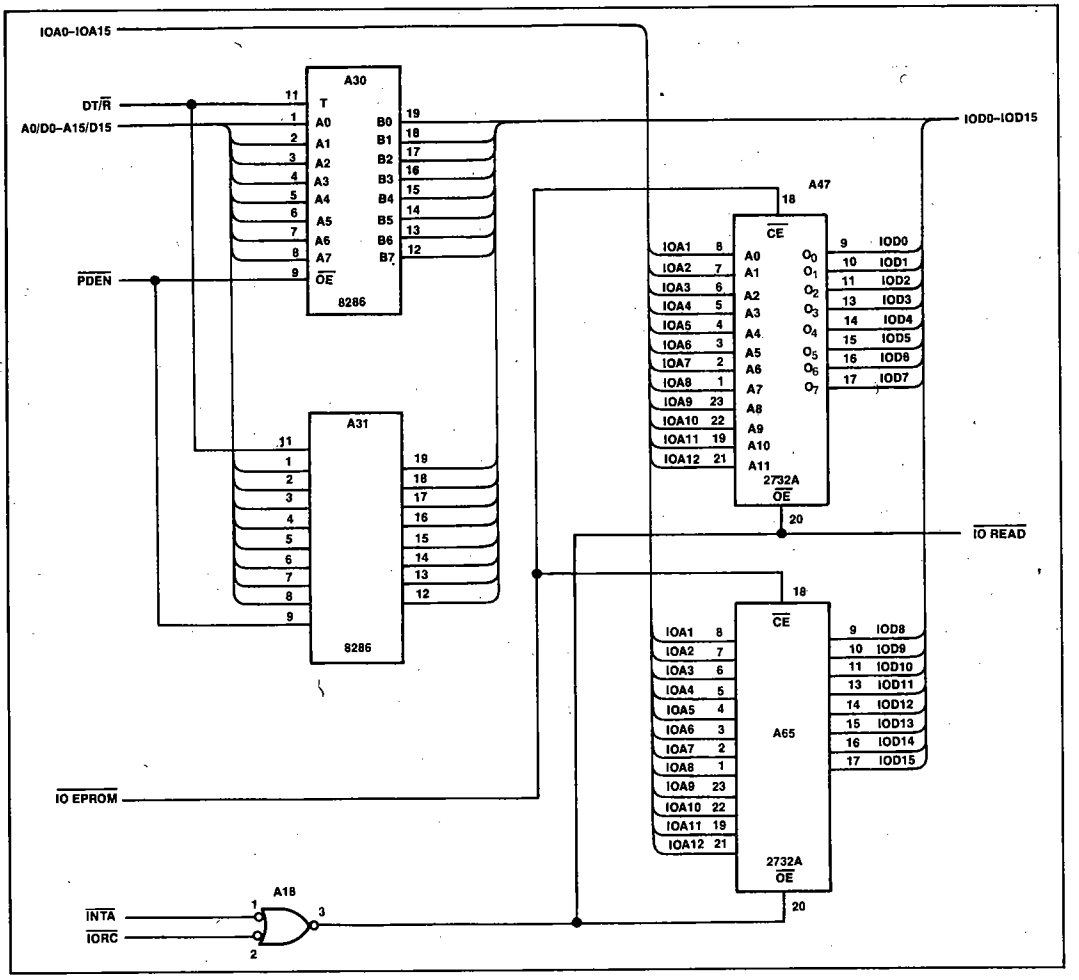


Figure 28. Bus Transceivers and EPROMs on I/O Bus

210355-001

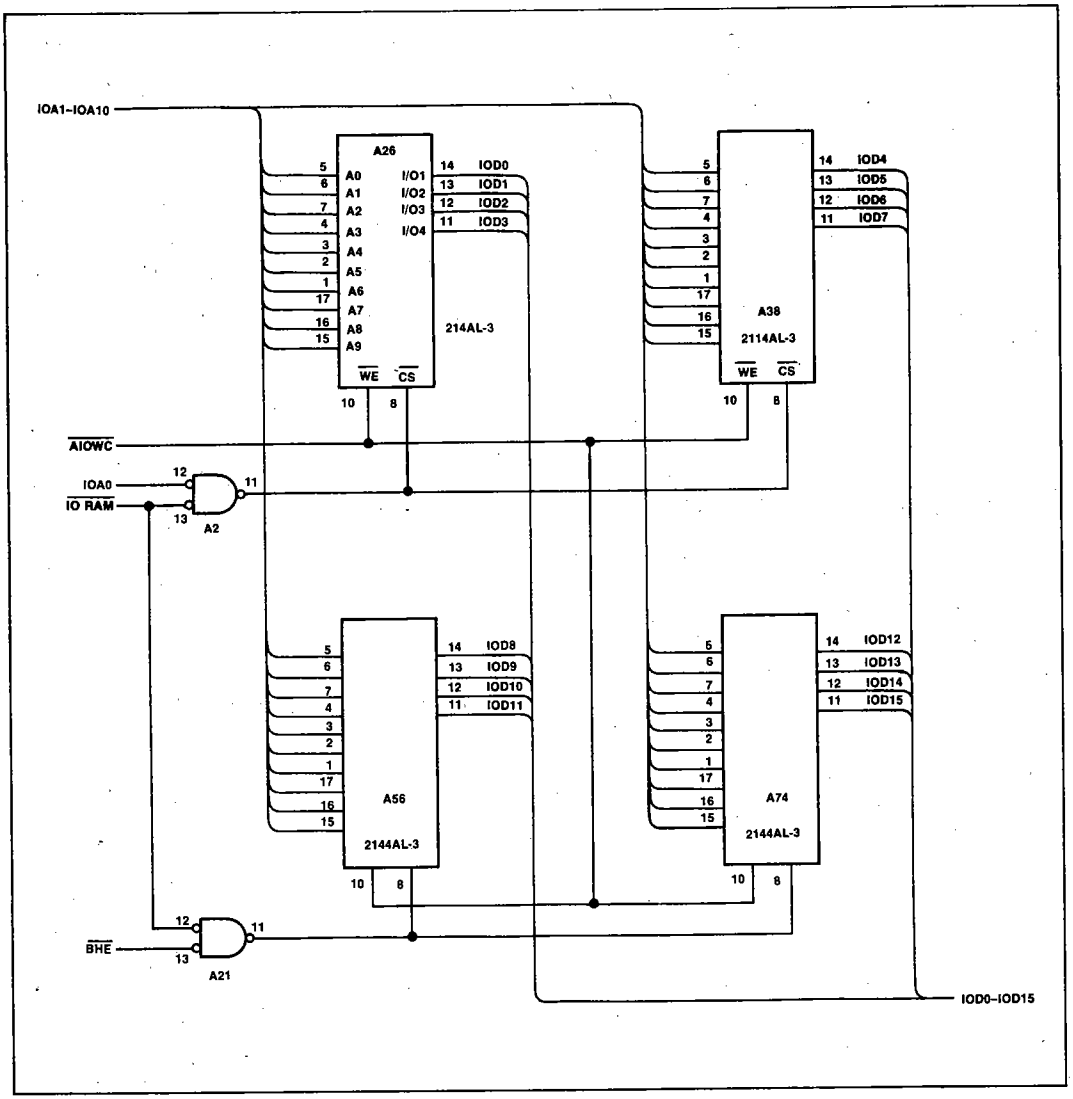


Figure 29. Static RAMs on I/O Bus

210355-001

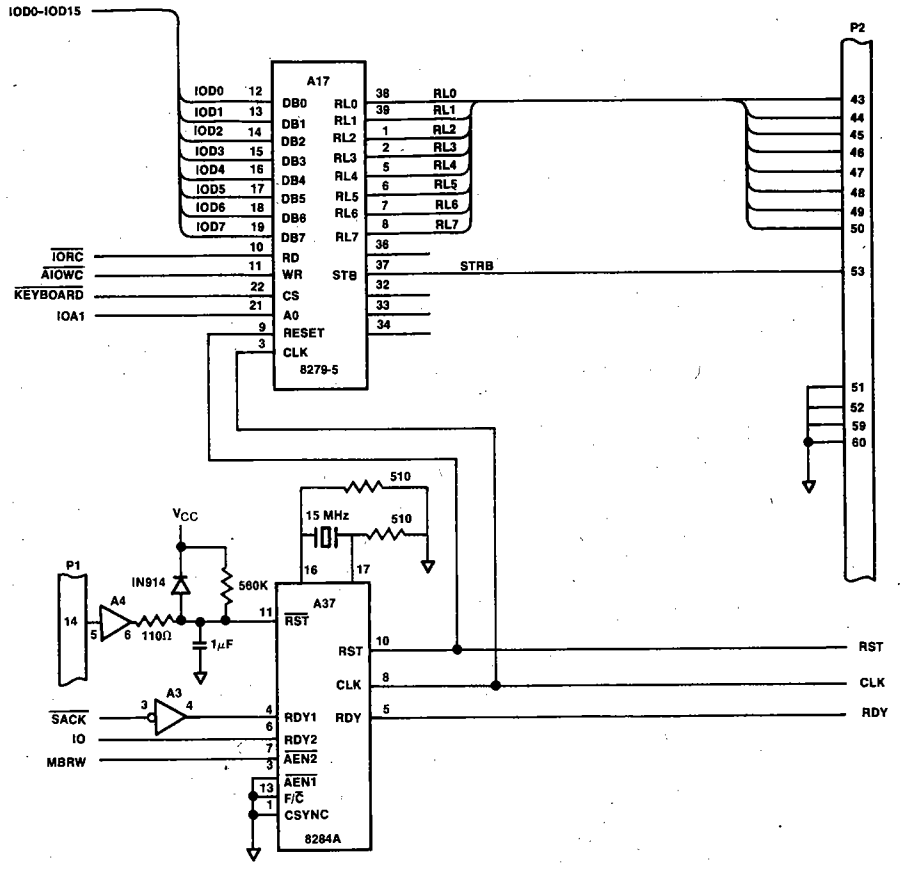


Figure 30. Keyboard Controller and Clock Generator

210355-001

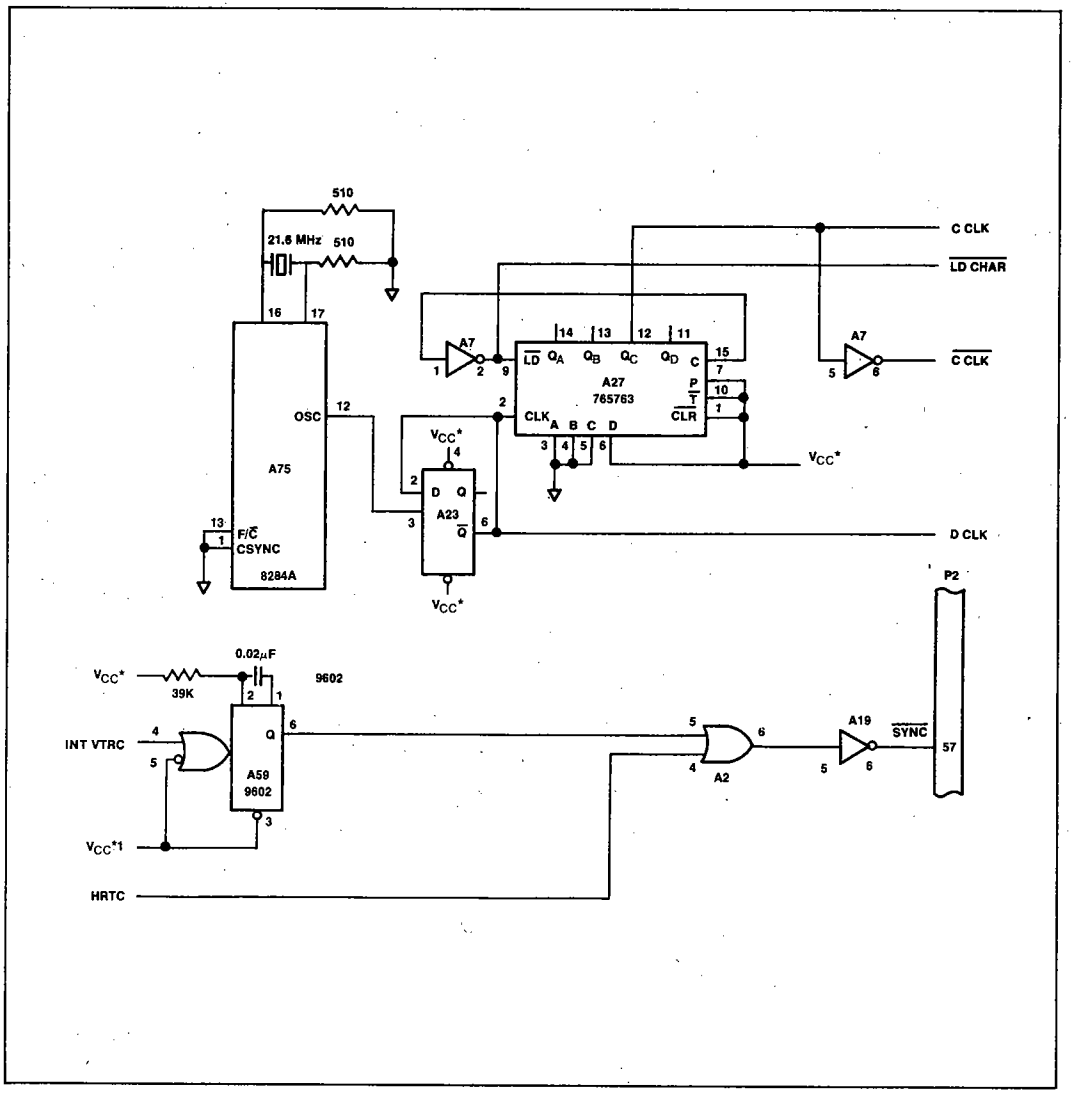


Figure 31. Character Clock Generator and Video Sync Pulse

210355-001

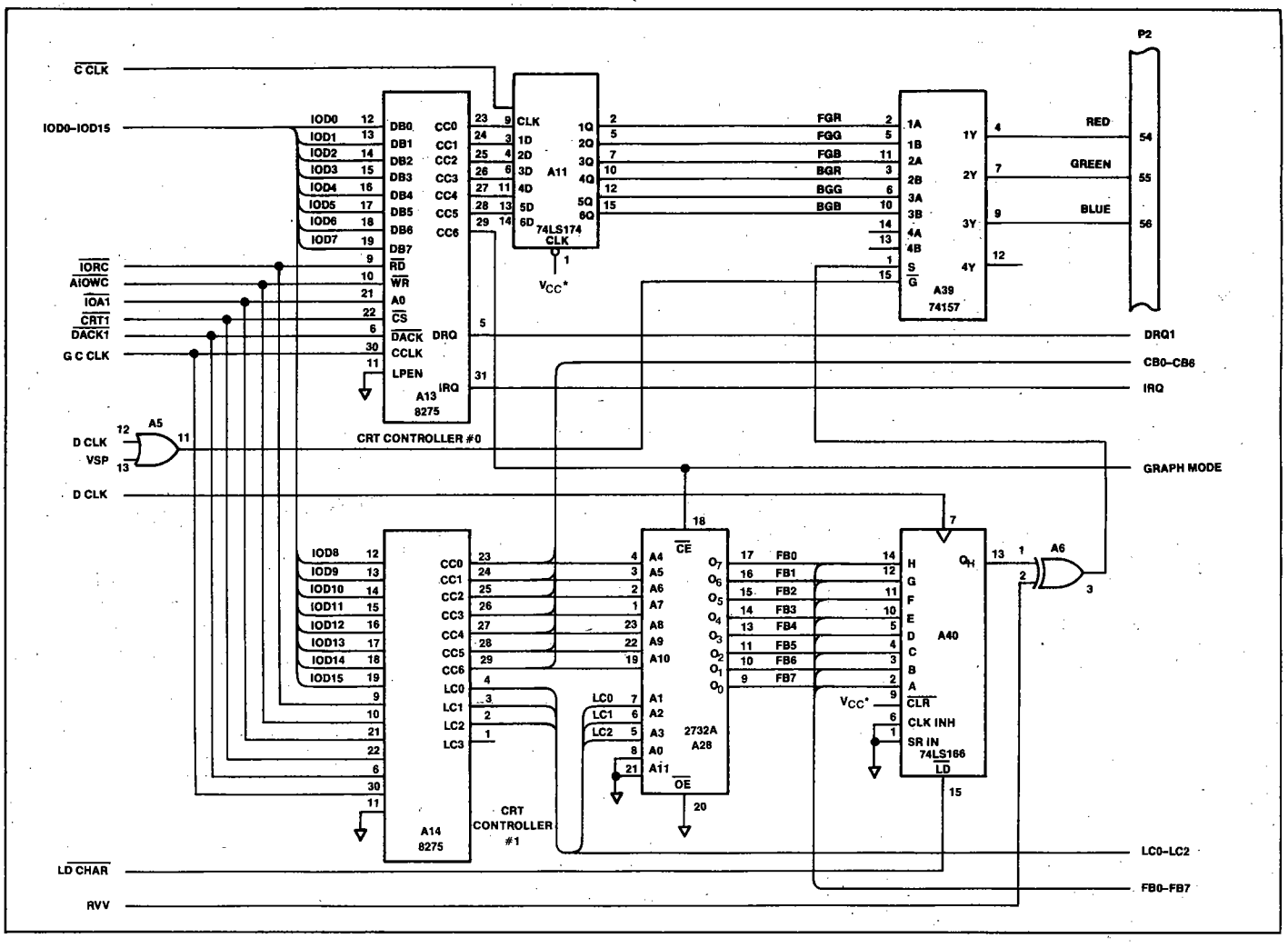


Figure 32. CRT Controllers, Color Multiplexer, and Character Generator

For each character, the foreground and background color bits are output from Byte 0 and latched into the 74LS174, from which they are applied to the input of the 74LS157 multiplexer. Selection between foreground and background is done by the output of the 74LS166 parallel-to-serial converter, which operates from either the text or graphic character generator, as appropriate. The roles of foreground and background color may be reversed by the RVV (reverse video) signal from the 8275, which is exclusive-ORed with this color select output.

Since the RBG (red-blue-green) inputs of the color monitor (Aydin Controls 8039D) are AC coupled, return-to-zero type outputs are needed to pass these signals through the input stages. This is provided by strobing the gate input of the 74LS157 multiplexer with the D CLK (dot clock) signal. By varying the duty cycle of the D CLK, the user can produce many different shades of color. The D CLK signal is ORed with the VSP (video suppress) signal from the 8275, to produce complete video blanking when desired.

Figure 33 shows the CRT Controllers #2 and #3, the decoder for the line select function, and latches for the video control signals. CRT controllers #2 and #3 are operational in graphics mode only. Synchronization of the two pairs of CRT controllers is discussed in the *8089 Display Functions Software* section.

Figure 34 shows the tri-state buffers used to handle the color information within a graphic character. The decoded line count outputs (ROW 0/–ROW 4/) are used to select which buffer is enabled onto the bus. The buffer A36, enabled by the GRAPH MODE signal, is used to "double up" the four graphic cells to produce eight (horizontal) dot inputs to the shift register (Figure 32).

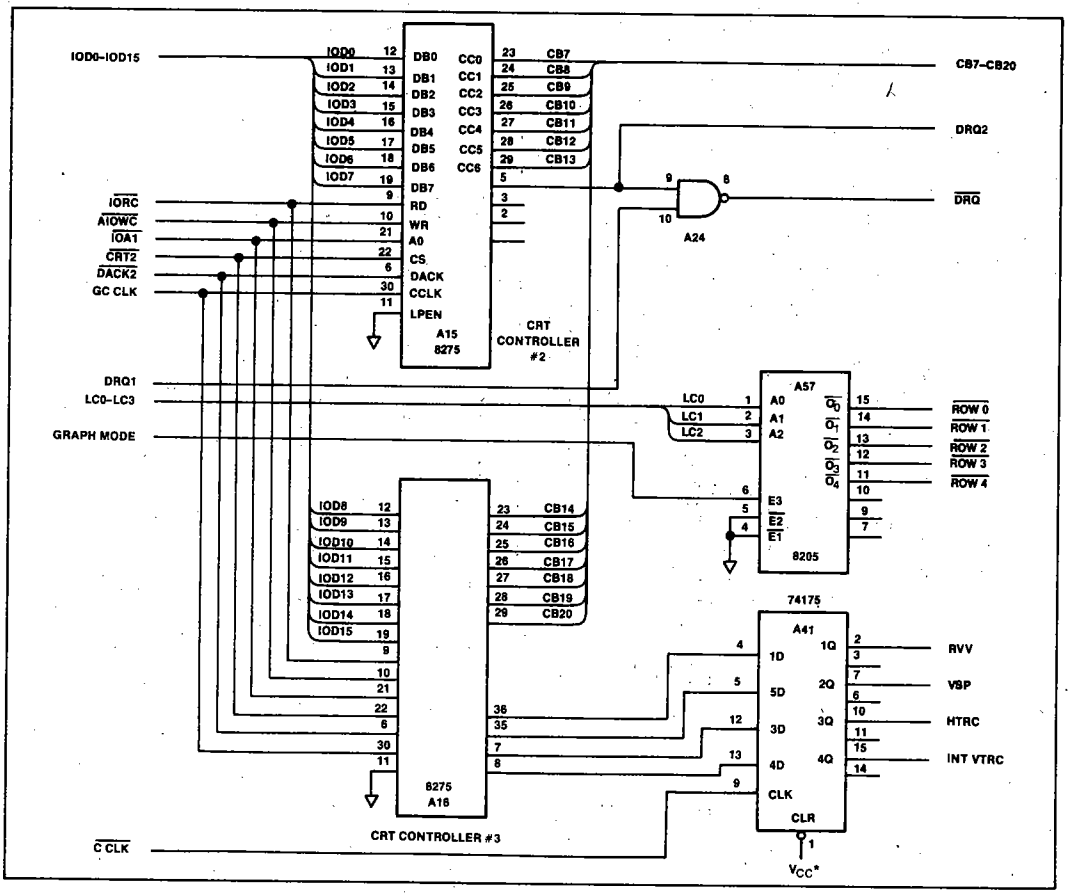


Figure 33. CRT Controllers, Line Decoder, and Video Control Signal Latch

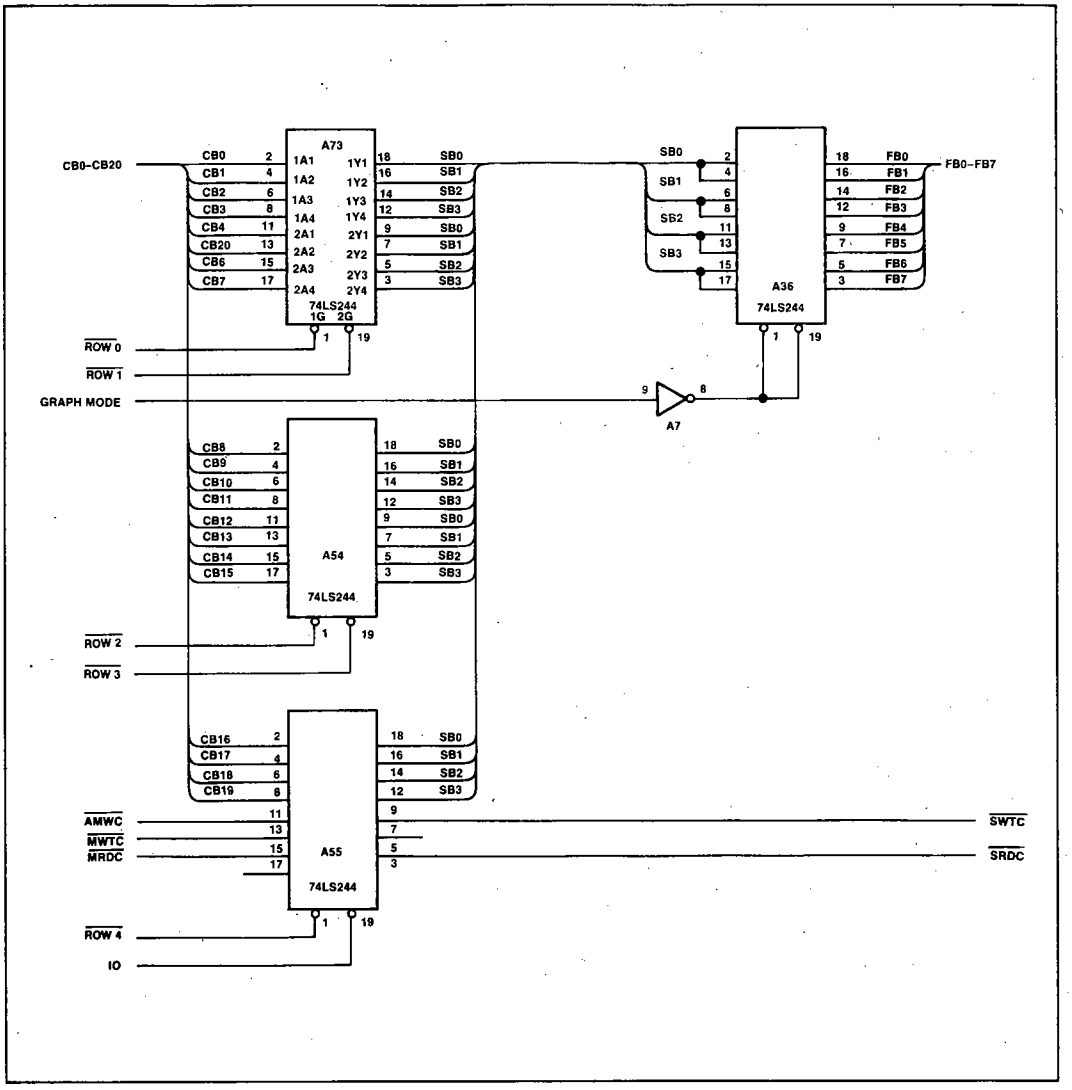


Figure 34. Tri-State Buffers for Graphic Color Information

The block diagram in Figure 35 shows how the text characters are processed. The following statements apply to Figure 35:

1. Byte 0, Bit 6 = 0 indicates text mode.
2. The six color signals from CRT Controller #0 (three foreground and three background) are latched and transmitted to the multiplexer.
3. The seven character output signals and the three line count signals from CRT Controller #1 are transmitted to the text character generator.
4. The eight output signals from the text character generator are transmitted to the parallel-to-serial converter.
5. The serial, horizontal dot data is transmitted to the multiplexer and selects foreground (dot data bit = 0) or background (dot data bit = 1) color signals.
6. The red, blue, and green color signals are transmitted to the color monitor.
7. CRT Controllers #2 and #3 are not operational in text mode.

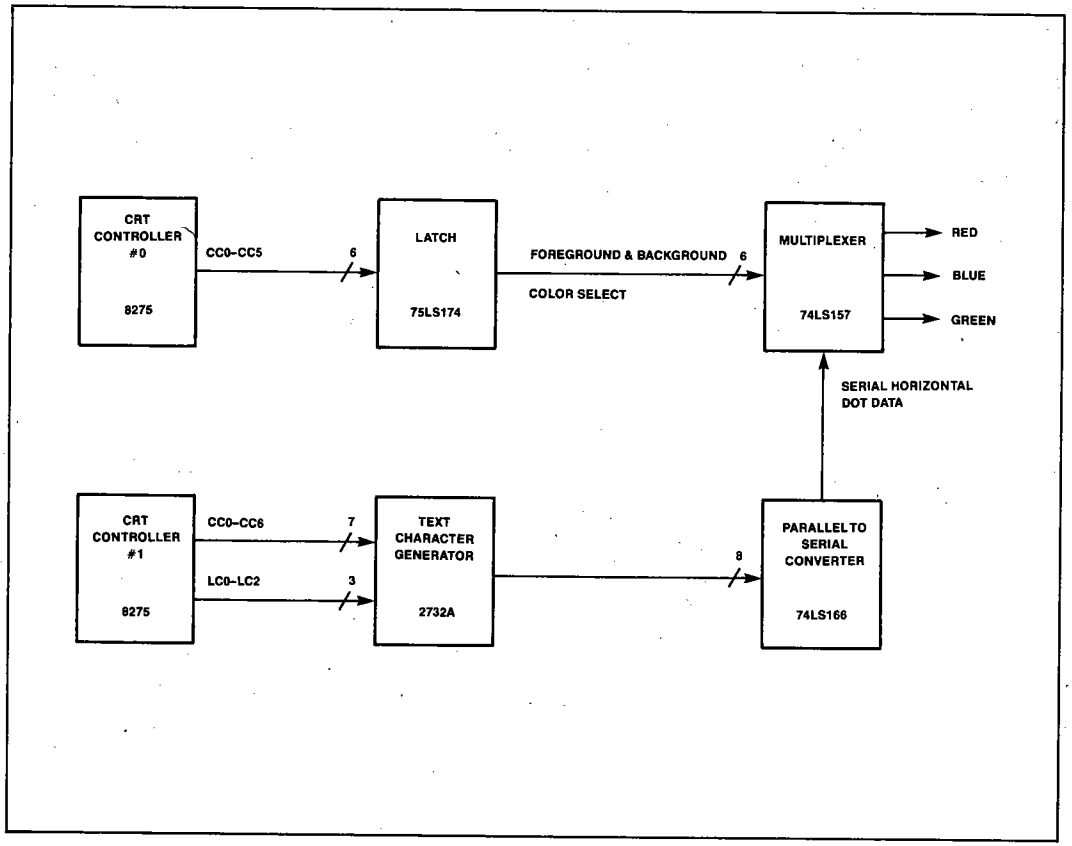


Figure 35. Processing of Text Characters

The block diagram in Figure 36 shows how graphic characters are processed. The following statements apply to Figure 36:

1. Byte 0, Bit 6 = 1 indicates graphic mode.
2. The six color signals from CRT Controller #0 (three foreground and three background) are latched and transmitted to the multiplexer.
3. The three line count signals from CRT Controller #1 are transmitted to a one-of-eight decoder which generates five row select signals (ROW 0–ROW 4).
4. The twenty pixel signals from CRT Controllers #1, #2, and #3 are transmitted to three octal buffers.
5. The four pixel signals of the selected row (based on the row select signals) are transmitted to another octal buffer.
6. The octal buffer converts these four bits to eight bits by duplicating each signal. Thus, output bits 0 and 1 are equal, 2 and 3 are equal, etc.
7. The eight output signals of the octal buffer are transmitted to the parallel-to-serial converter.
8. The serial, horizontal dot data is transmitted to the multiplexer and selects foreground (dot data bit = 0) or background (dot data bit = 1) color signals.
9. The red, blue, and green color signals are transmitted to the color monitor.

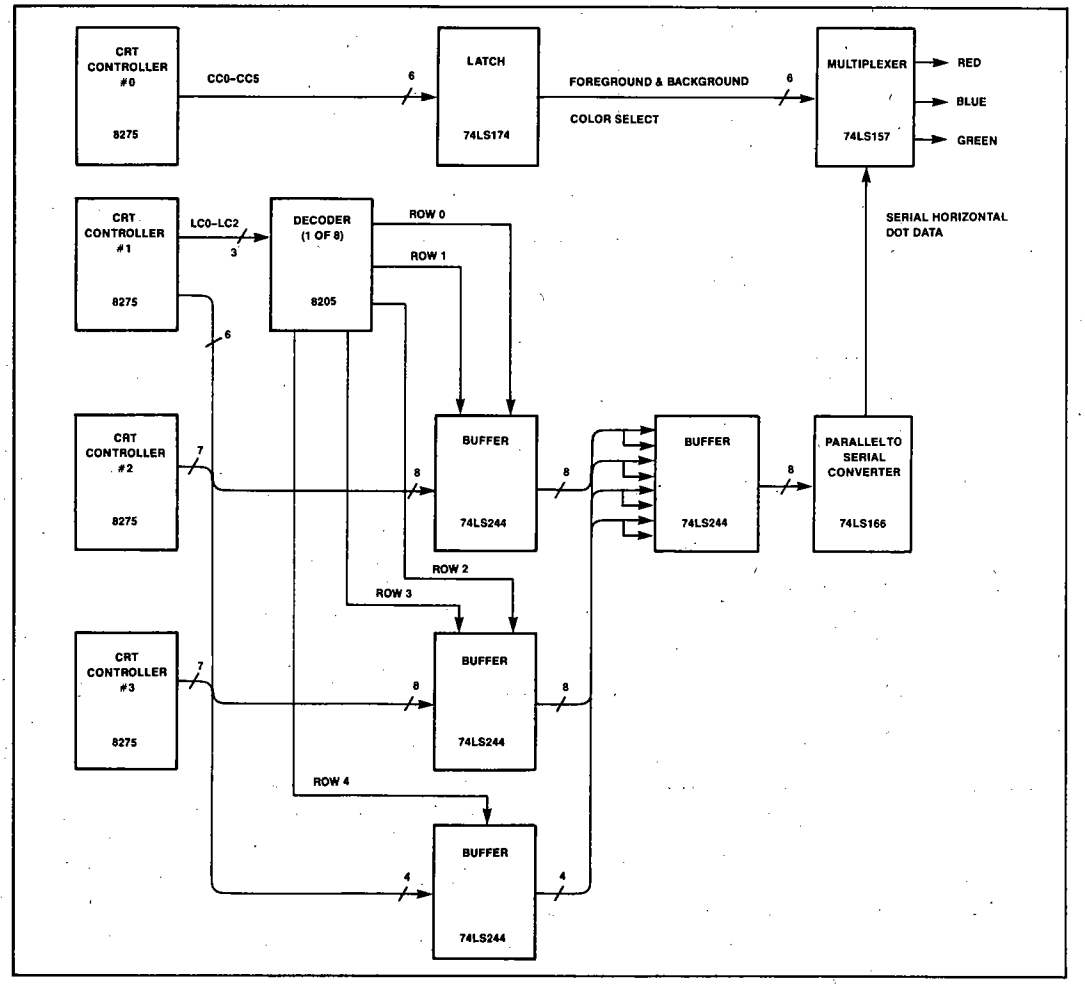


Figure 36. Processing of Graphic Characters

Figure 37 shows the circuit used to synchronize the 8275s, and also the circuit used to generate the DRQF signal. As mentioned earlier (see Figure 20), if the 8089 were to wait for a subsequent DRQ signal from the 8275s, some clock cycles would be allocated to idle clocks, and the DMA transfer would become less efficient. To preclude this, the circuit shown in Figure 37 generates a surrogate (early) DRQ signal, DRQF, using a one-shot triggered by the trailing edge of DRQ (DRQ 1 AND DRQ 2). The one-shot times out prior to the rising edge of CLK in T4 of the DMA's store bus cycle.

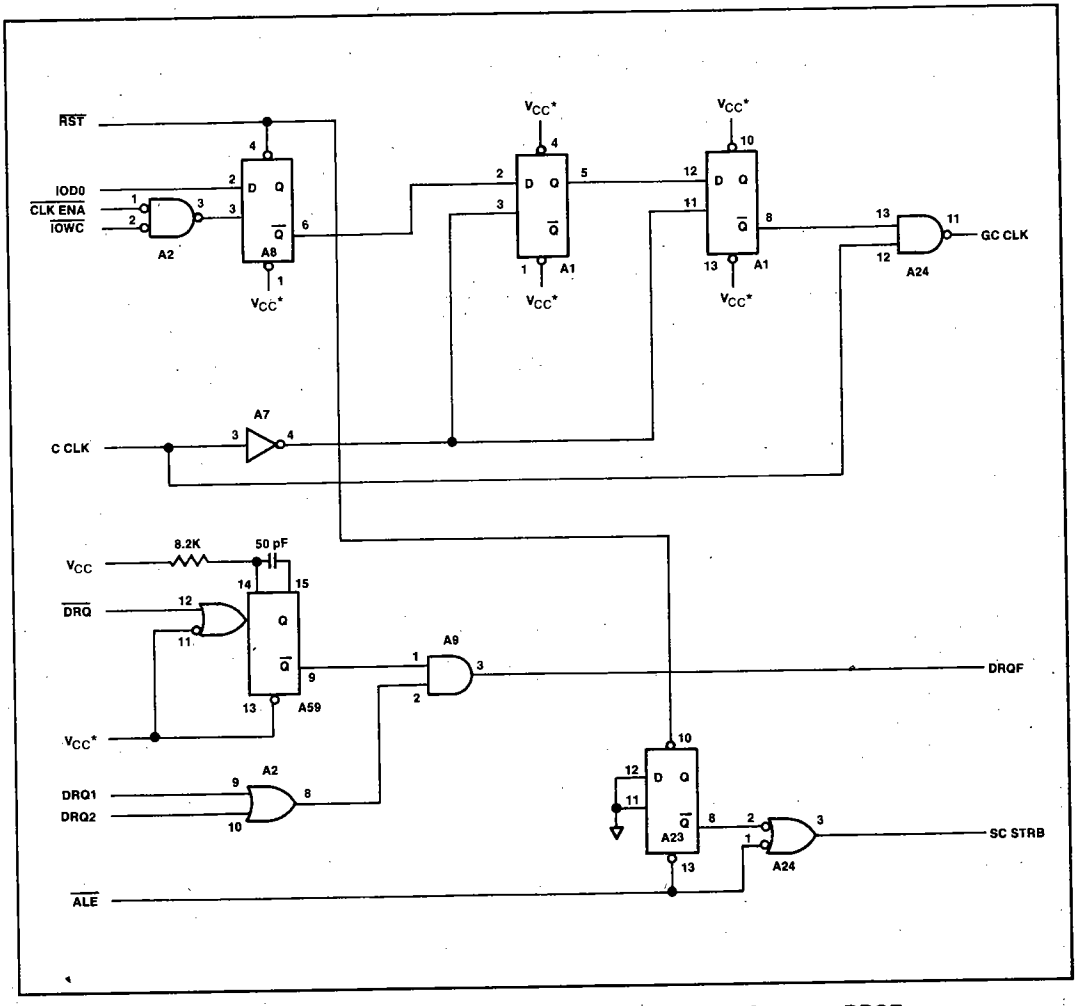


Figure 37. Circuits to Synchronize CRT Controllers and Generate DRQF

Figure 38 shows the relationship between the individual DRQ signals from the 8275s and the DRQF signal that is sent to the 8089. DRQ 1 is the data request representing the 8275s #0 and #1, while DRQ 2 similarly represents the 8275s #2 and #3. The DACK 1/ and DACK 2/ signals (along with AIOWC/) are used to deactivate DRQ 1 and DRQ 2, respectively.

Figure 39 shows the multiplexer used to control writing of data to the dual-port RAM. When IO and SWTC/ are both low, the 8089 data is gated to the dual-port RAM. When BDSEL/ and SWTC/ are both low, the 8086 data is gated to the dual-port RAM. BDSEL/ may be active only when the 8089 is in the I/O space. Note that the address range for the dual-port RAM is F8000–FFFFF as seen by the 8089, and F0000–F7FFF as seen by the 8086.

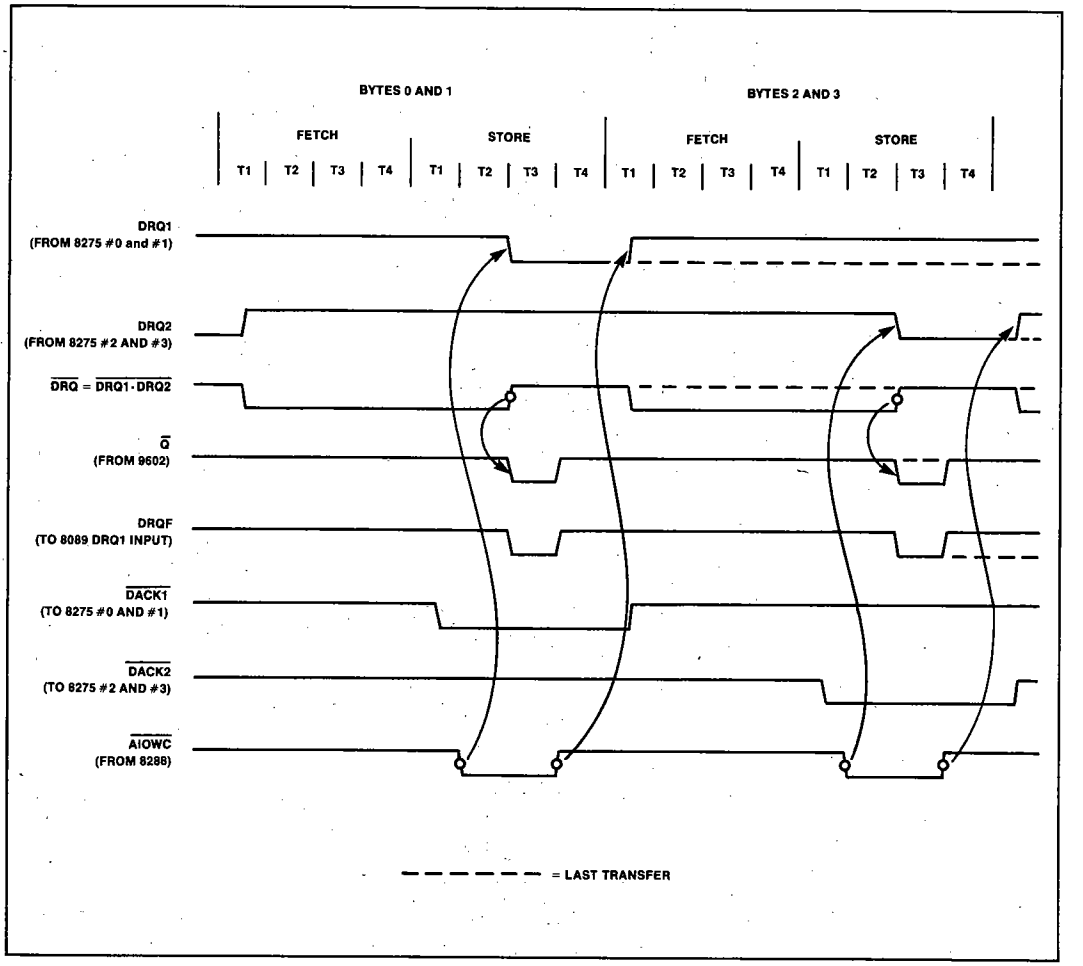


Figure 38. Derivation of DRQF Signal

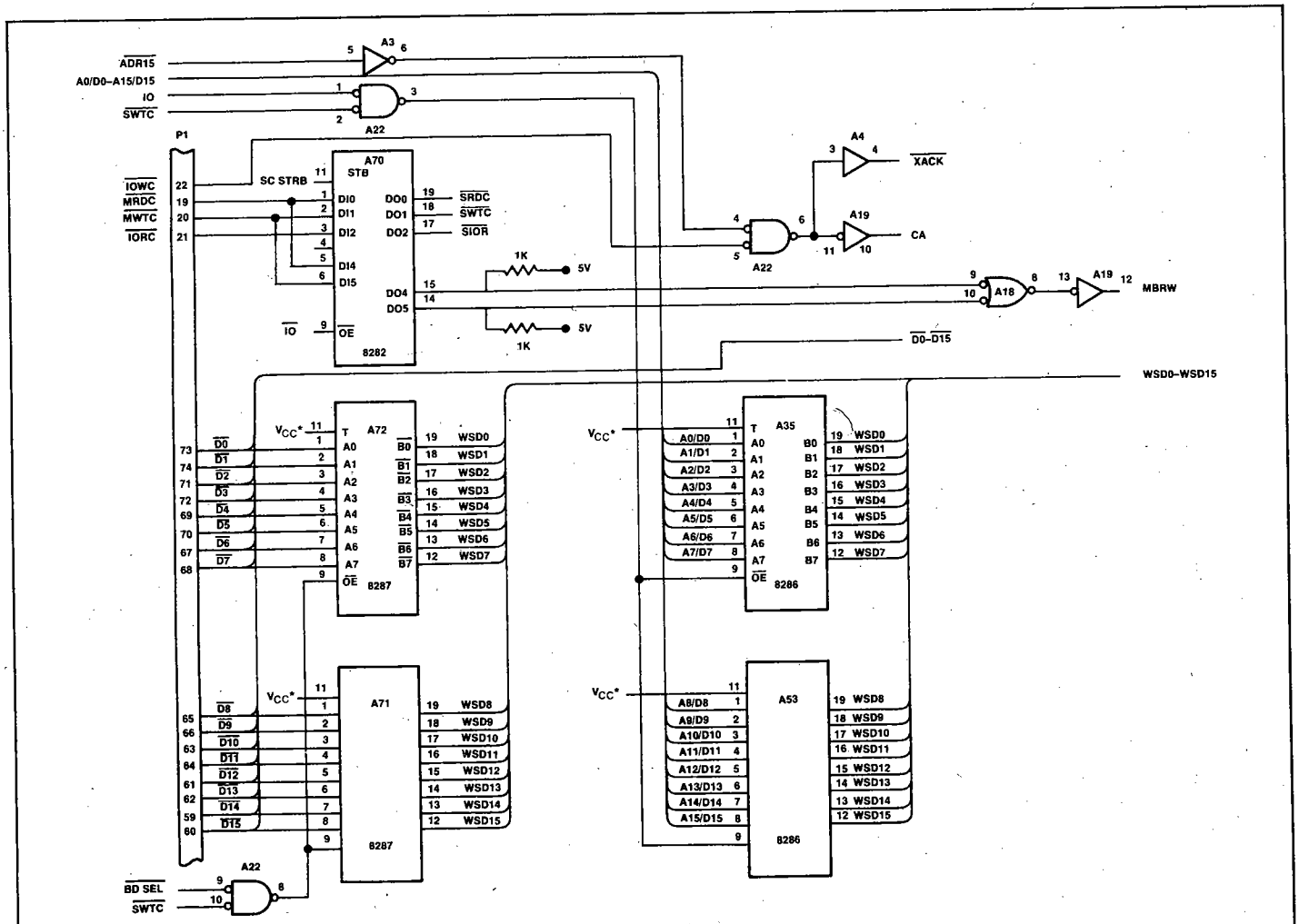


Figure 39. Multiplexer for Writing to Dual-Port RAM

Figure 40 shows the demultiplexer used to control reading of data from the dual-port RAM. The internal transfer acknowledge (SACK/) signal from the dynamic RAM controller latches this data. If MRDC/ is active, the data is then gated to the 8089. If BD ENA/ is active, the data is gated to the Multibus for transmission to the 8086.

Figure 41 shows the multiplexer for the address inputs to the dual-port RAM. If the IO signal is high, the address on the Multibus is gated into the dual-port RAM. If IO is low, the address from the 8089 is gated into the dual-port RAM.

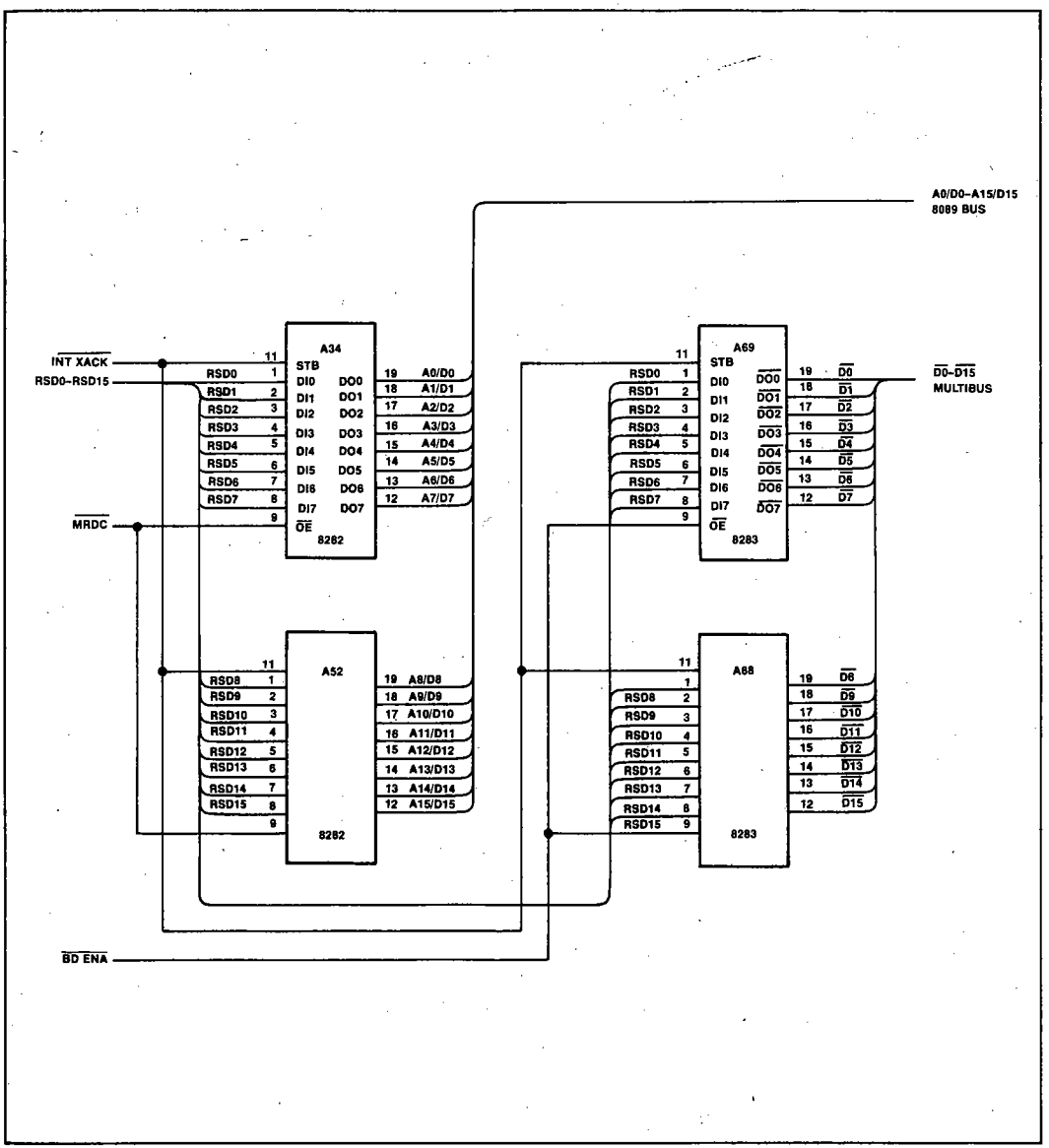


Figure 40. Demultiplexer for Reading from Dual-Port RAM

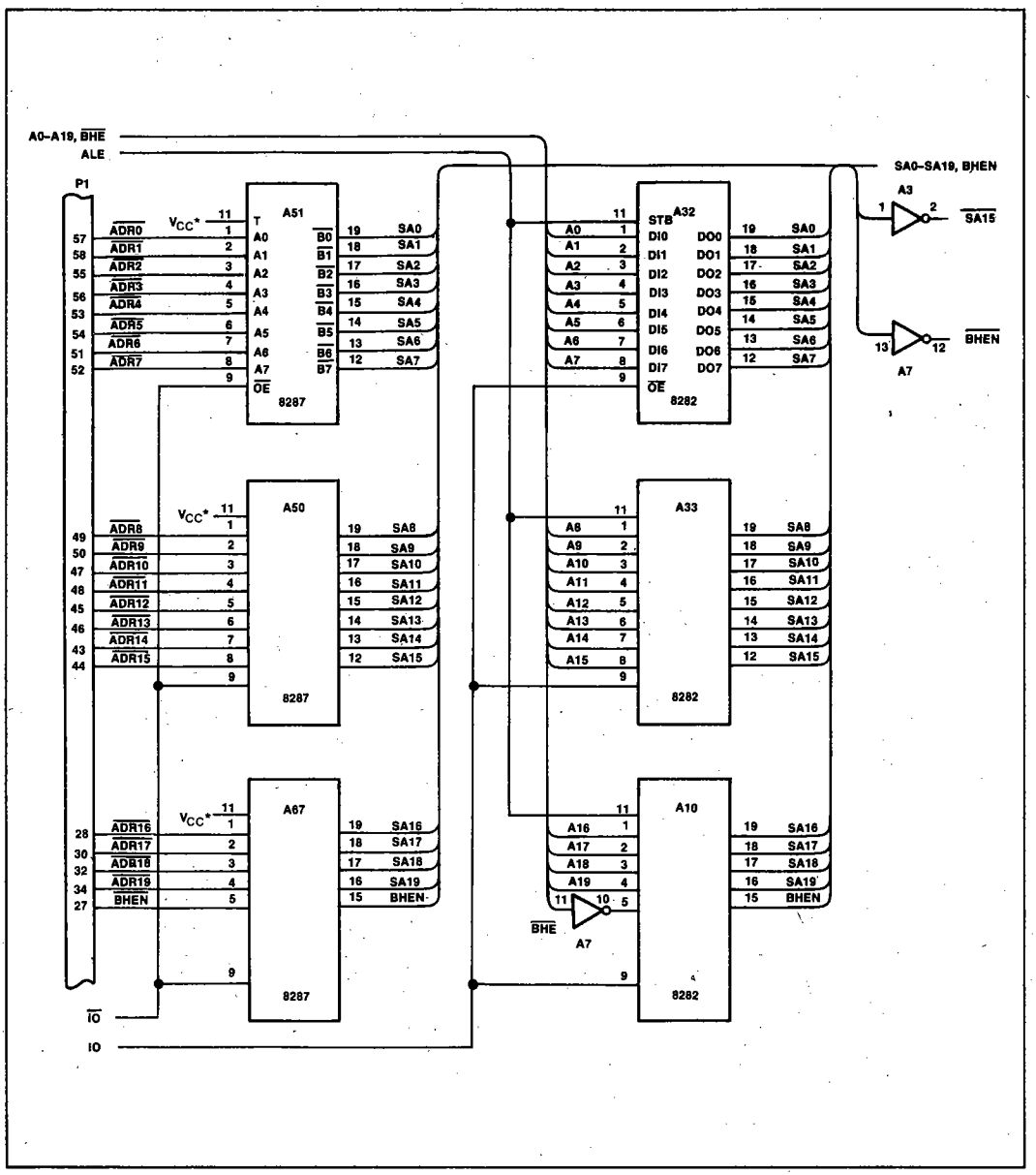


Figure 41. Multiplexer for Address Inputs to Dual-Port RAM

Figure 42 shows the 8202 dynamic RAM controller. The inputs SA0–SA19 come from the multiplexer shown in Figure 41. The dynamic RAM controller generates the control signals (shown at the right of the page) for operating the dynamic RAM.

Figures 43 and 44 show the dynamic RAM itself.

8089 Display Functions Software

The 8089 display functions software consists of a single program which is executed by the 8089 on a continuous basis. This program performs the following functions:

Initialization for the 8089 itself and for the CRT controllers and the keyboard controller.

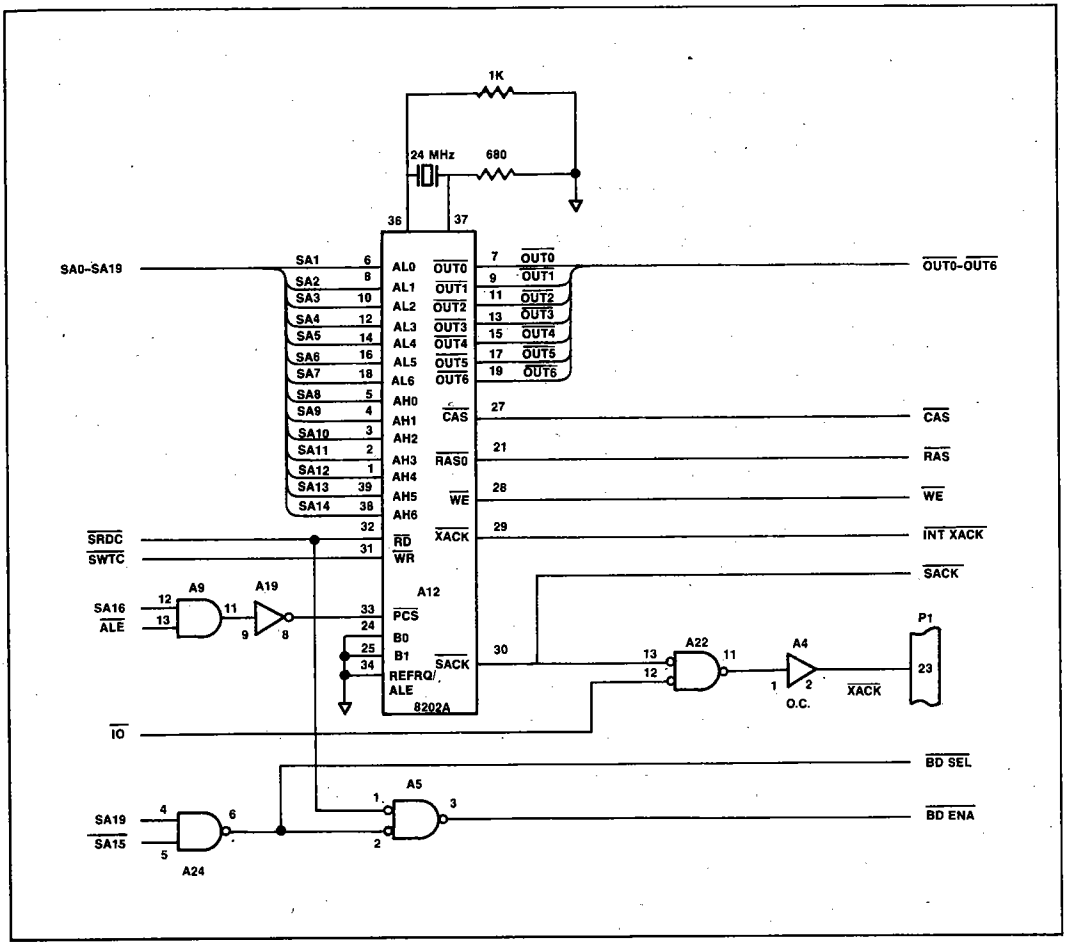


Figure 42. Dynamic RAM Controller

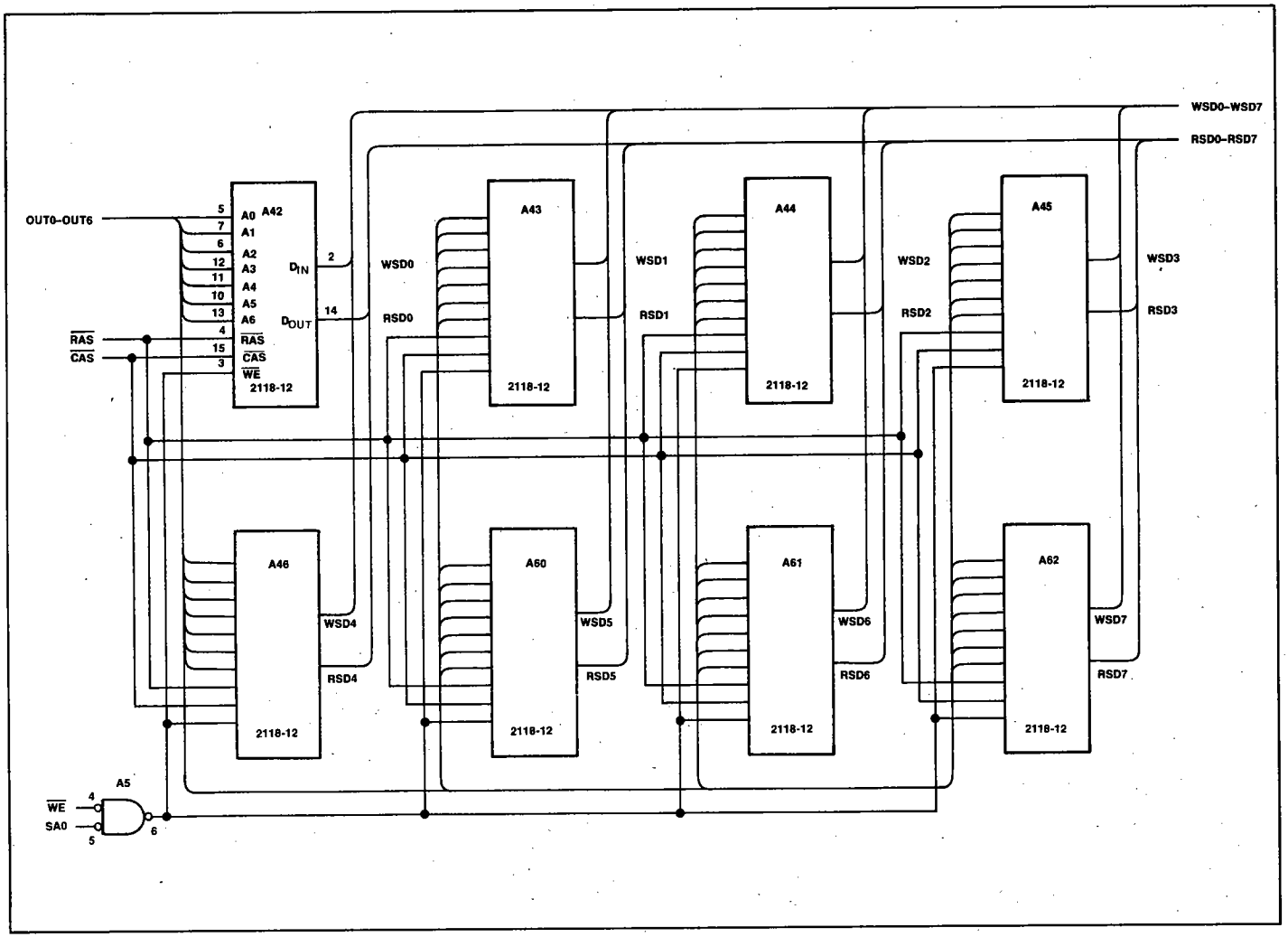


Figure 43. Dynamic RAM (Low Data Byte)

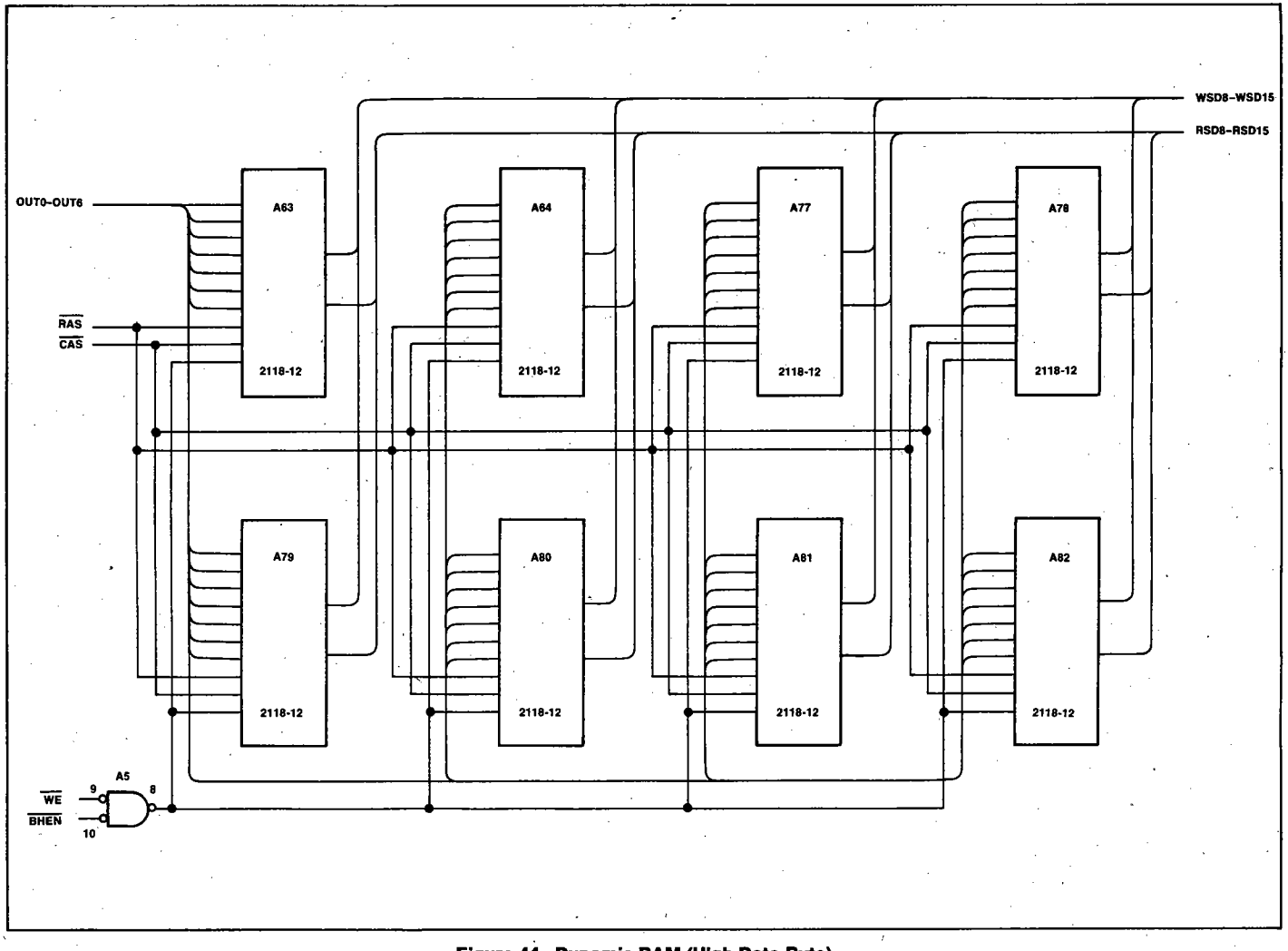


Figure 44. Dynamic RAM (High Data Byte)

The transfer instruction which causes the DMA transfer of the CRT refresh data to begin.

Polling routines for the keyboard and the command buffer.

Figure 45 is a simplified flowchart showing the relationships among these three main functions. The program begins upon receipt of the second CA (channel attention) following an IOP reset. After the initialization processes have been completed, the program loops continuously, alternating between DMA transfer and polling processes. There are 48 rows of characters on the screen. The polling processes are carried out during the vertical retrace time, which is the equivalent of 2 rows. Thus, it is easy to see that the DMA process uses up 96% of the 8089's time, leaving 4% for the polling processes.

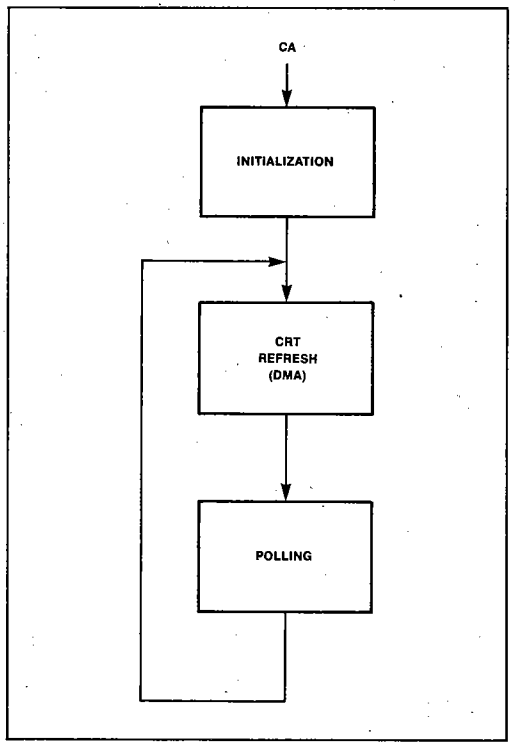


Figure 45. Channel Program Simplified Flowchart

As mentioned earlier, the channel program is stored in the 2732A EPROMs on the I/O bus. Figure 23 (above) shows the address assignments for devices on the I/O bus. The 2732As occupy addresses 2000–3FFF. The 8089 also uses a scratch-pad static RAM (2K bytes at addresses 0000–07FF). The CRT controllers are accessed by using addresses 4000 and 6000 on the I/O bus. Address 6000 is "CRT Controller 1" and actually refers to the first pair of 8275s. Address 4000 is "CRT Controller 2," the second pair of 8275s. Address 8000 is a clock enable address. Write commands to this address enable or disable the GC clock, which is the character clock for the 8275s. Address A000 is decoded to produce the DACK signal for the 8275s. Address C000 is the address of the keyboard controller.

The exact manner in which the channel program executes depends on the flag settings and parameter values in the parameter block.

Appendix A is a flowchart for the complete channel program. Appendix B is the corresponding ASM-89 assembly language listing. In the paragraphs to follow, a general overview of the channel program is given. The reader may refer to the flowchart and listing if a more detailed description is desired.

The first CA after IOP reset causes the 8089 to fetch the system configuration pointer (SCP) and system configuration block (SCB) from dual-port memory. These blocks contain certain very basic system-level information for the 8089, as explained above under *Overview of the 8089*.

The next CA causes the channel program to begin execution (at the point marked START on the flowchart). The initialization portion of the channel program consists of the following operations:

Start and initialize the 8275 CRT controllers.

Initialize the 8279 keyboard controller.

Initialize the dual-port variables (parameter block).

Synchronize the 8275 CRT controllers.

To initialize and synchronize the 8275s, the channel program performs the following operations:

Enable the GC CLK to the 8275s by writing 01H to I/O port address 8000H.

Send the Reset command to the 8275s, followed by the four screen format parameters (all commands sent to the 8275s are sent first to the pair of 8275s at address 6000H and then repeated for the second pair of 8275s at address 4000H).

Send the Preset Counters command to the 8275s.

Disable the GC CLK by writing 00H to address 8000H.

Send the Start Display command to the 8275s.

Enable the GC CLK again by writing 01H to address 8000H. The 8275s are now initialized and synchronized.

After the initializations have been completed, the channel program enters its main loop. The 8089 channel control register is loaded to specify the following DMA conditions:

Data transfer from memory to I/O port.

Destination-synchronized transfer.

GA register pointing to data source.

Termination on external event.

Termination offset = 0.

The source for the DMA transfer (display page 0 or 1) is then selected according to the value of DSPLY_PG_PTR (the display page pointer initialized by the host CPU) in the parameter block. The CRT character clock is then started and the DMA transfer begins. When the entire screen has been refreshed, the 8275s activate the 8089's EXT input.

The 8089 then executes the SINTR instruction, which causes an interrupt to be sent to the 8086 (SINTR-1 line on the Multibus), to notify the 8086 that the page transfer has been completed. The 8089 then reads the CRT controller status registers which causes the IRQ signal (from the 8275s to the 8089) to be reset.

The channel program then begins the polling process which checks for ASCII commands from the 8086 (in the command buffer) and also for key depressions at the keyboard. In addition to the alphanumeric characters, the channel program recognizes the following control characters:

Character	Code	Description
CNTRL-A	01	Monitor Inhibit
CNTRL-B	02	Monitor Uninhibit
CNTRL-C	03	EEPROM,Inhibit
CNTRL-D	04	EEPROM Uninhibit
CNTRL-E	05	Turn on EEPROM Buffer
CNTRL-F	06	Display Page 0
CNTRL-G	07	Display Page 1
CNTRL-H	08	Backspace
CNTRL-I	09	TAB (Every 8 Characters)
CNTRL-J	0A	Linefeed
CNTRL-K	0B	EEPROM Buffer Off
CNTRL-L	0C	Erase Page
CNTRL-M	0D	Carriage Return
CNTRL-N	0E	Set Background Color
CNTRL-O	0F	Set Foreground Color
CNTRL-P	10	Set Color to Black
CNTRL-Q	11	Set Color to Red
CNTRL-R	12	Set Color to Green
CNTRL-S	13	Set Color to Yellow
CNTRL-T	14	Set Color to Blue
CNTRL-U	15	Set Color to Magenta
CNTRL-V	16	Set Color to Cyan
CNTRL-W	17	Set Color to White
CNTRL-X	18	Abort Line
CNTRL-Y	19	Cursor Right
CNTRL-Z	1A	Cursor Down and Left
CNTRL-^	1E	Cursor Up
CNTRL-/	1C	Cursor Home
CNTRL-DEL	1F	Recall EEPROM Buffer

The first four commands listed above are not recognized if they originate from the physical keyboard, but are recognized if they appear as ASCII commands in the command buffer (that is, if they come from the 8086). Refer to the flowchart (Appendix A) for more details on how the channel program responds to the control characters.

System Performance

The 8089 performs DMA transfers on 921,600 bytes of display data per second. In addition, the 8089 executes a polling routine (described above) during the vertical retrace time (the equivalent of two display rows). The DMA transfer (for a single frame) takes 16.000 milliseconds. This leaves .667 milliseconds for the polling routine to execute, out of a total of 1/60-second CRT refresh period. The program listed in Appendix B takes about 300 microseconds to execute, approximately half the available time. When the polling process is finished, the channel program goes back to DMA mode, and waits for the first DRQ signal from the 8275s.

While the polling routine is executing, the 8089 makes most of its memory accesses in the I/O space, and the dual-port RAM is available to the 8086. When the 8089 returns to the DMA routine, however, it hangs the dual-port RAM while waiting for DRQ. This occurs because the fetch from the dual-port RAM deactivates the IO signal which locks out the 8086 from the dual-port RAM. The IO signal is then not activated until DRQ is received and the data is written to the CRT controllers. This can adversely affect system throughput. Therefore, if it is desired to increase the 8086's access to the dual-port RAM during this period, the user should insert NOPs into the channel program so that it spends more time in the I/O space before returning to DMA.

The 8086 may also access dual-port RAM during the DMA transfer. The dual-port RAM is available to the 8086 on approximately a 50% duty cycle (during the store portion of the DMA transfer cycle). The 8089's store cycle is 800 nanoseconds long (assuming a 5 MHz clock). The 8086's access to dual-port RAM (assuming an 8 MHz clock) takes 500 nanoseconds. However, since the two processors operate asynchronously, the 8086 may begin its access at any point during the 8089's

DMA store cycle. Since the 8086 is the master relative to the dual-port RAM, the ready signal for the 8089's next fetch operation will not be generated until the 8086 is through. Thus, on occasion, the 8089 will have to wait.

Each row of characters requires 256 microseconds of DMA transfer time if no such wait states occur. The repetition rate for rows of characters is 333 microseconds (1/3000 second). Thus, the accumulated wait states due to the 8086's access to dual-port RAM may total 77 microseconds before any underrun occurs. The 8086 programs should be written in such a manner that the added wait states do not total 77 microseconds during any one period of 333 microseconds. The most important single factor in assuring this is to avoid making long burst transfers to or from the dual-port RAM. If an underrun does occur, the entire screen will be blanked until the beginning of the next frame.

Aside from the shared access to dual-port RAM, the two processors may operate concurrently with no coordination necessary. Operations performed by the 8086 (such as numeric processing of display data) may be programmed without regard to the overhead associated with IOP operations.

Conclusions

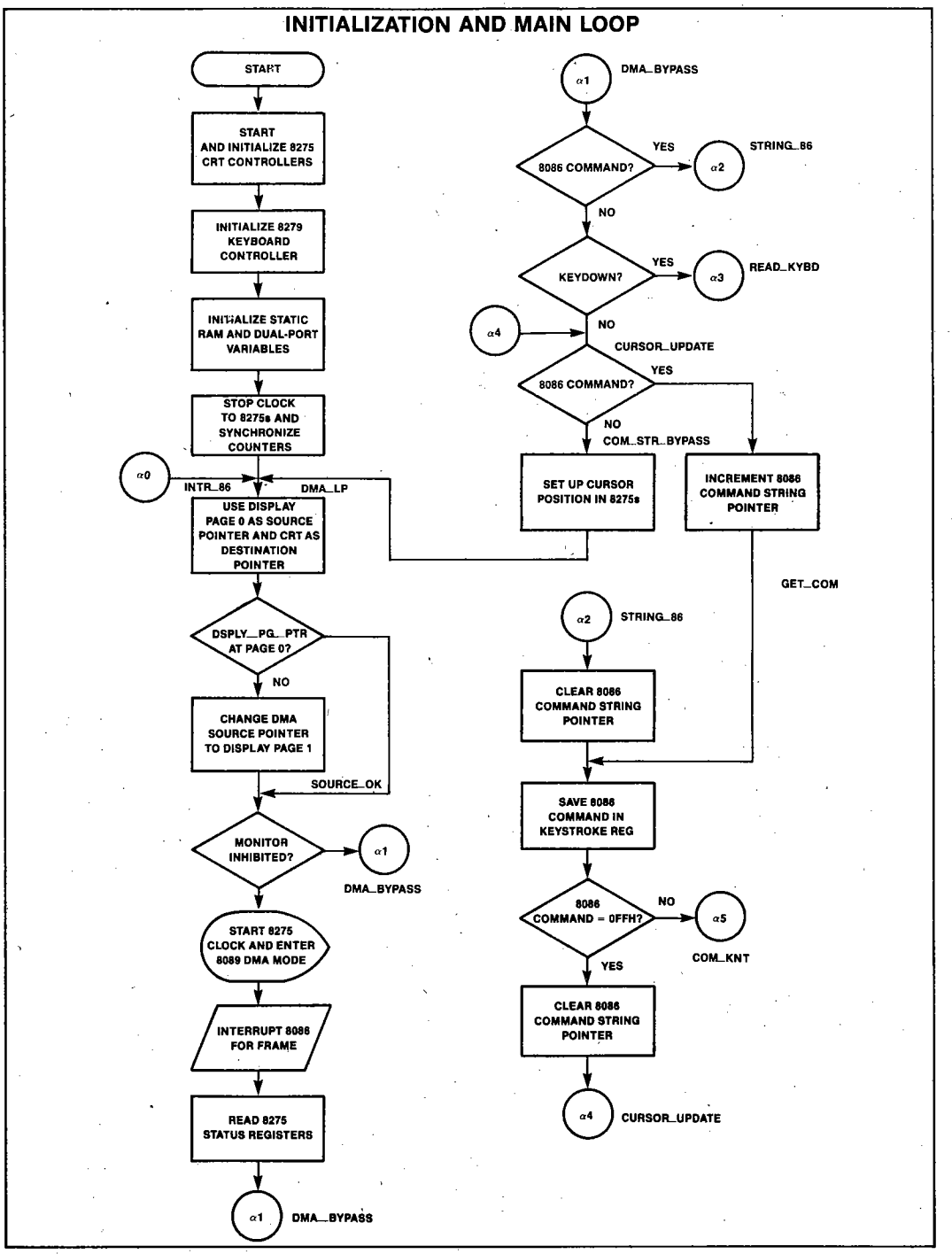
This application note has demonstrated that a high-performance, color-graphic CRT terminal can be conveniently built using the Intel 8089 microprocessor system. This system utilizes a high-performance 8086 CPU operating at 8 MHz and an 8089 I/O processor operating at 5 MHz.

In particular, the unique abilities of the 8089 lend themselves to the graphic CRT application by enabling a true multiprocessing approach to be used. The following list summarizes the capabilities used in this specific design:

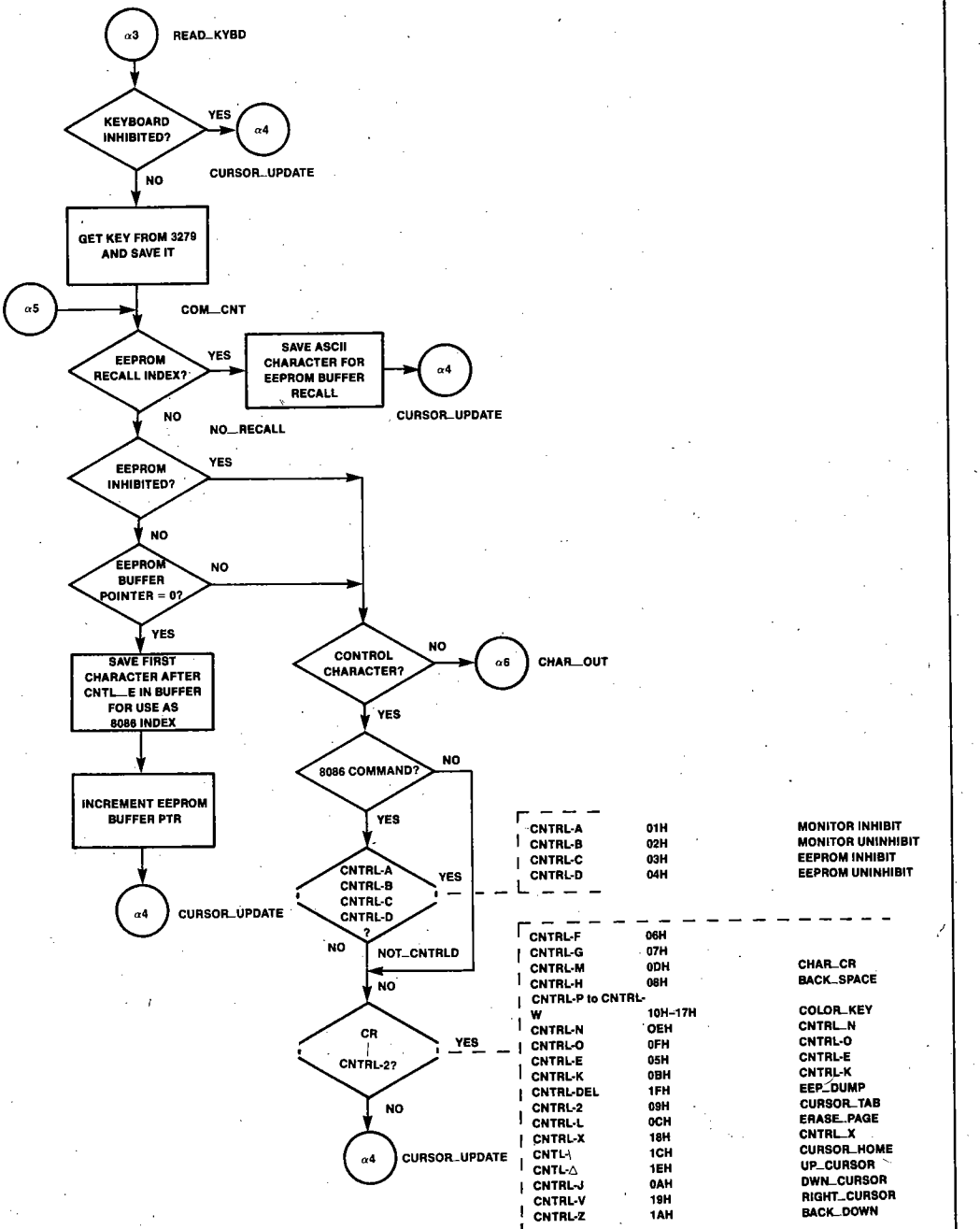
- High-speed DMA transfers (up to 1.25 megabytes/second) without wait states.
- Capabilities of a CPU and a DMA controller in a single 40-pin package.
- Support of concurrent operation for the system CPU and the I/O processor. Ability to access memory and address devices on both a system bus and a separate I/O bus.
- Flexible, memory-based communications between the I/O processor and the system CPU.
- Capability for 1-megabyte addressing in the system space.
- Capability for 16-bit DMA transfer, with external event termination.
- Support of modular, subsystem development effort due to the simple software interface (memory-based communications, plus channel attention and interrupt signals) and the simple hardware interface (CA, SEL, and SINTR lines).

The following 8089 capabilities were not used in the design described in this note, but may be useful in other graphic CRT systems or I/O processing systems:

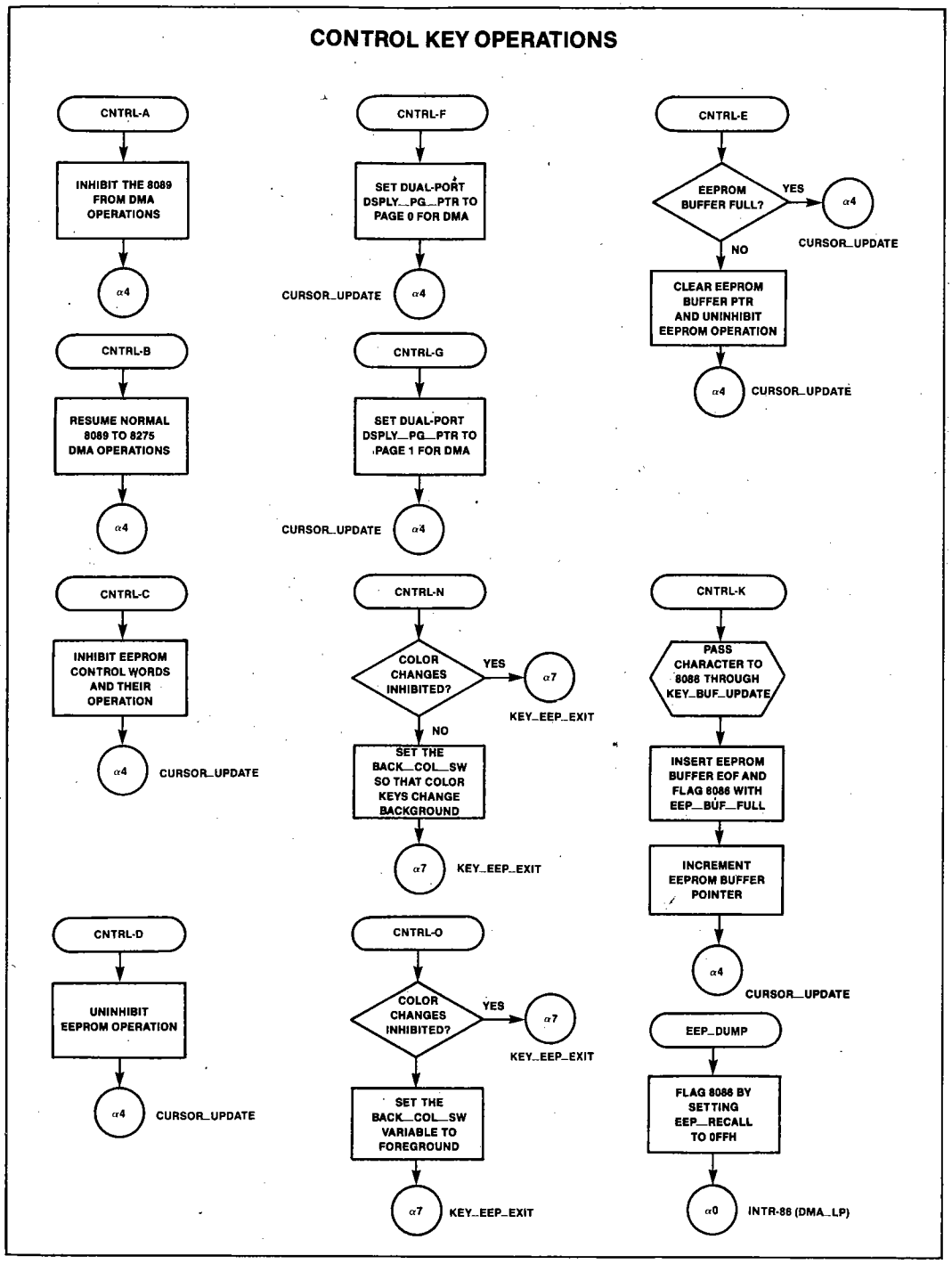
- Two channels, each of which may execute instructions and perform DMA transfers.
- Bit manipulation instructions.
- Support of both 8-bit and 16-bit bus width in the system space and in the I/O space.
- Enhanced DMA capabilities, including:
  - Translation (e.g., ASCII to EBCDIC code).
  - Termination on masked compare.
  - Word assembly/disassembly (8-bit word to/from 16-bit word).
  - Memory-to-memory or I/O-to-I/O transfer.
  - Synchronization on source, destination, or neither.


INITIALIZATION AND MAIN LOOP
START
START AND INITIALIZE 8275 CRT CONTROLLERS
INITIALIZE 8279 KEYBOARD CONTROLLER
INITIALIZE STATIC RAM AND DUAL-PORT VARIABLES
STOP CLOCK TO 8275s AND SYNCHRONIZE COUNTERS
α0
INTR_86
DMA_LP
USE DISPLAY PAGE 0 AS SOURCE POINTER AND CRT AS DESTINATION POINTER
DSPLY_PG_PTR AT PAGE 0?
NO
CHANGE DMA SOURCE POINTER TO DISPLAY PAGE 1
SOURCE_OK
MONITOR INHIBITED?
α1
DMA_BYPASS
START 8275 CLOCK AND ENTER 8089 DMA MODE
INTERRUPT 8086 FOR FRAME
READ 8275 STATUS REGISTERS
α1
DMA_BYPASS
α1
DMA_BYPASS
8086 COMMAND?
YES
α2
STRING_86
NO
KEYDOWN?
YES
α3
READ_KYBD
α4
NO
CURSOR_UPDATE
8086 COMMAND?
YES
NO
COM_STR_BYPASS
SET UP CURSOR POSITION IN 8275s
INCREMENT 8086 COMMAND STRING POINTER
GET_COM
α2
STRING_86
CLEAR 8086 COMMAND STRING POINTER
SAVE 8086 COMMAND IN KEYSTROKE REG
8086 COMMAND = 0FFH?
NO
α5
COM_KNT
YES
CLEAR 8086 COMMAND STRING POINTER
α4
CURSOR_UPDATE

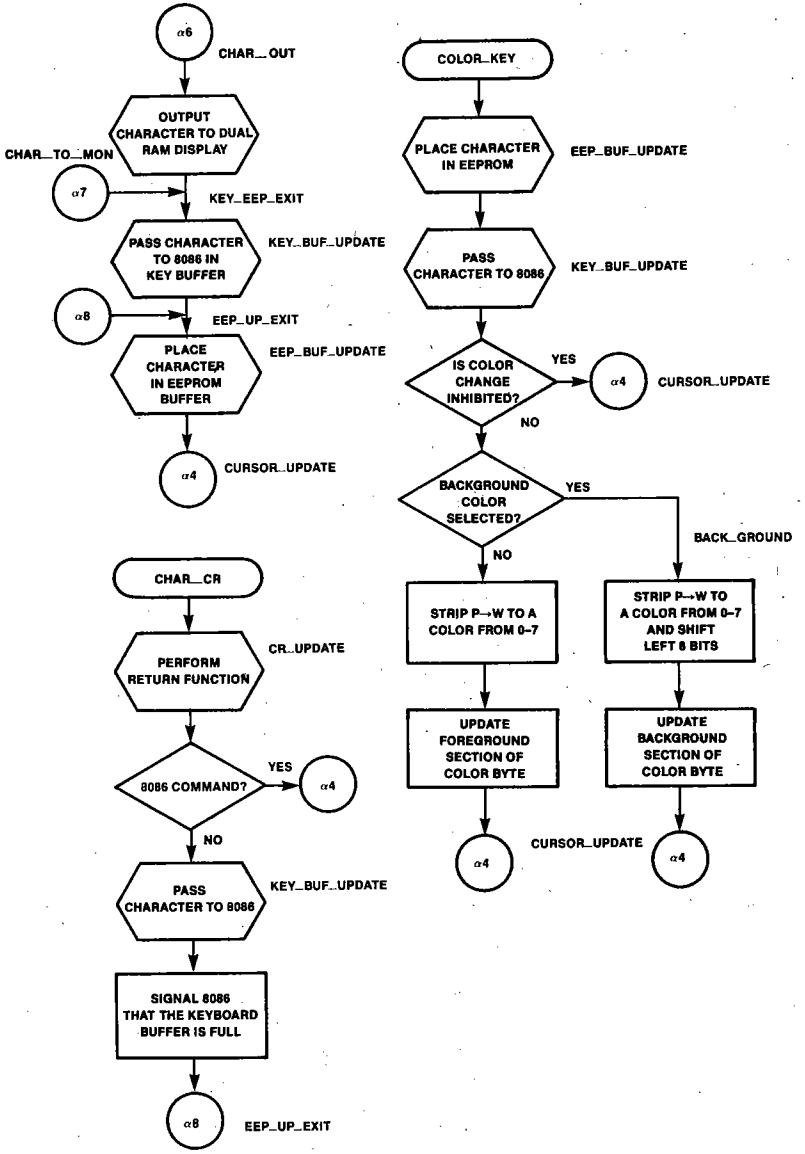
KEY AND COMMAND DECODE



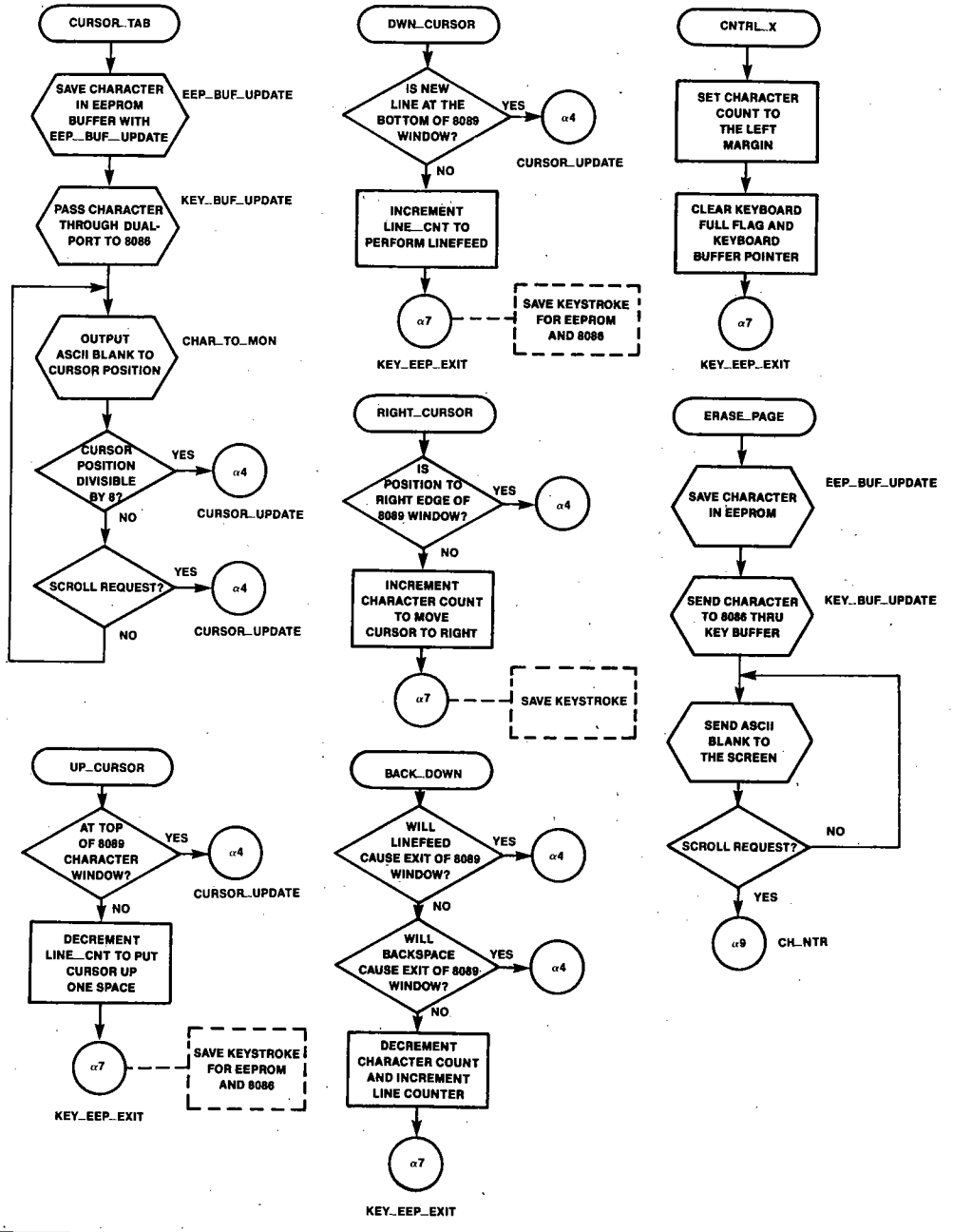
210355-001


CONTROL KEY OPERATIONS
CNTRL-A
INHIBIT THE 8089 FROM DMA OPERATIONS
α4
CNTRL-B
RESUME NORMAL 8089 TO 8275 DMA OPERATIONS
α4
CNTRL-C
INHIBIT EEPROM CONTROL WORDS AND THEIR OPERATION
α4
CURSOR_UPDATE
CNTRL-D
UNINHIBIT EEPROM OPERATION
α4
CURSOR_UPDATE
CNTRL-F
SET DUAL-PORT DSPLY_PG_PTR TO PAGE 0 FOR DMA
CURSOR_UPDATE
α4
CNTRL-G
SET DUAL-PORT DSPLY_PG_PTR TO PAGE 1 FOR DMA
CURSOR_UPDATE
α4
CNTRL-N
COLOR CHANGES INHIBITED?
YES
α7
KEY_EEP_EXIT
NO
SET THE BACK_COL_SW SO THAT COLOR KEYS CHANGE BACKGROUND
α7
KEY_EEP_EXIT
CNTRL-O
COLOR CHANGES INHIBITED?
YES
α7
KEY_EEP_EXIT
SET THE BACK_COL_SW VARIABLE TO FOREGROUND
α7
KEY_EEP_EXIT
CNTRL-E
EEPROM BUFFER FULL?
YES
α4
CURSOR_UPDATE
NO
CLEAR EEPROM BUFFER PTR AND UNINHIBIT EEPROM OPERATION
α4
CURSOR_UPDATE
CNTRL-K
PASS CHARACTER TO 8086 THROUGH KEY_BUF_UPDATE
INSERT EEPROM BUFFER EOF AND FLAG 8086 WITH EEP_BUF_FULL
INCREMENT EEPROM BUFFER POINTER
α4
CURSOR_UPDATE
EEP_DUMP
FLAG 8086 BY SETTING EEP_RECALL TO 0FFH
α0
INTR-86 (DMA_LP)

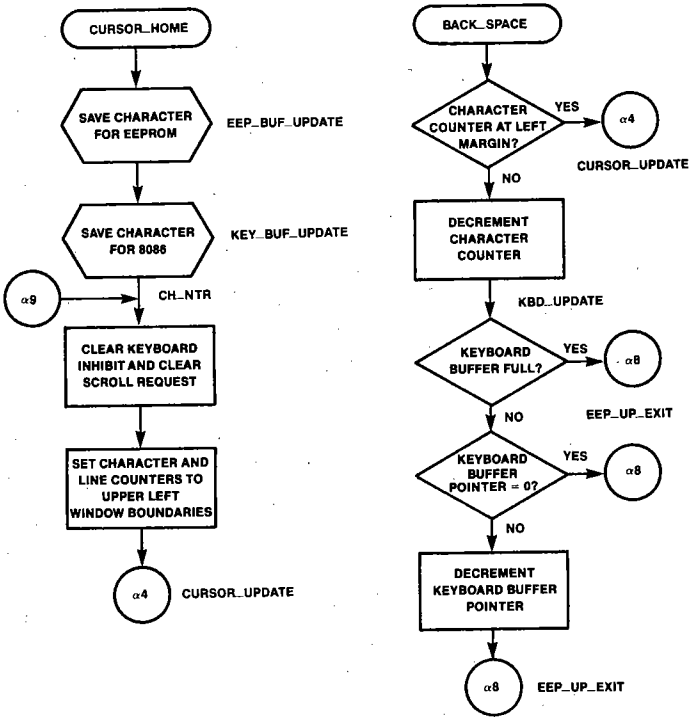
CONTROL KEY OPERATIONS



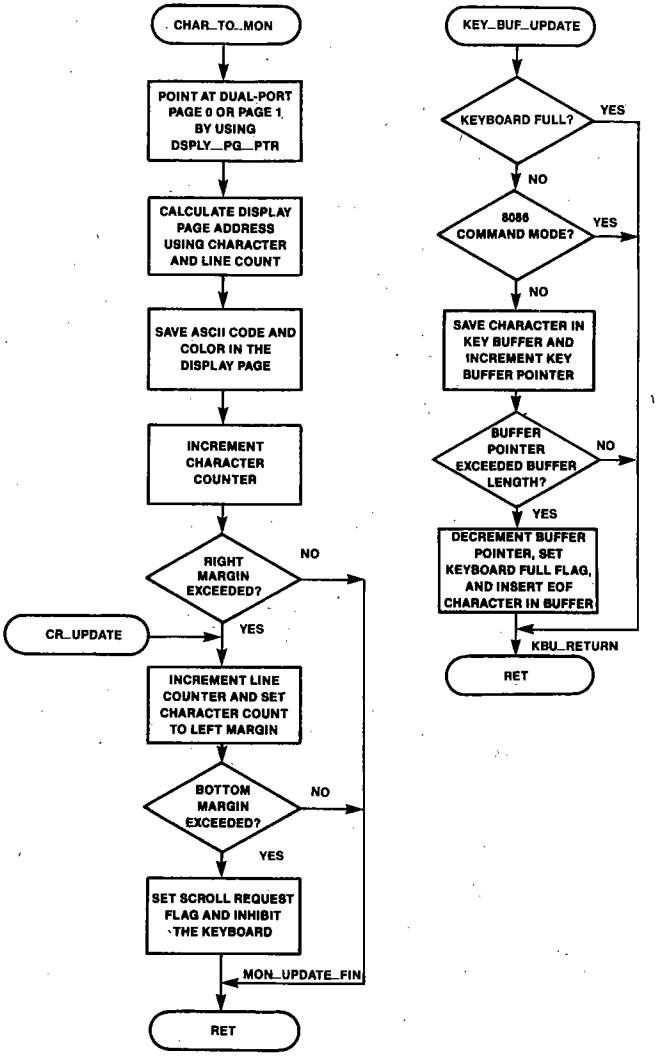
CONTROL KEY OPERATIONS



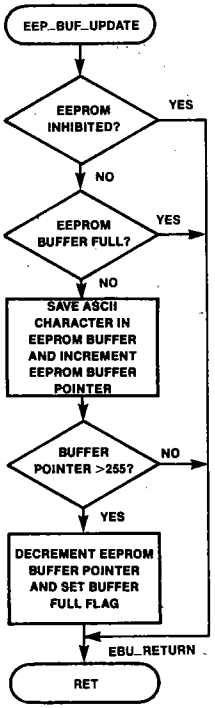
CONTROL KEY OPERATIONS



SUPPORT SUBROUTINES



SUPPORT SUBROUTINES



```
8089 MACRO ASSEMBLER


ISIS-II 8089 MACRO ASSEMBLER X202 ASSEMBLY OF MODULE N89
OBJECT MODULE PLACED IN :F1:N89.OBJ
ASSEMBLER INVOKED BY:   :F2:ASM89   :F1:N89.SRC

  1 ;
  2 ;   8089 DUMB TERMINAL PROGRAM
  3 ;
  4 ;   B.K.NELSON
  5 ;
  6 ;   STARTED:   4/30/80
  7 ;   LAST CHANGE:  8/12/80
  8 ;
  9 ;   THIS PROGRAM INITIALIZES FOUR 8275 CRT CONTROLLERS AND A
 10 ;   8279 KEYBOARD CONTROLLER.   ASCII INFORMATION FLOW MAY FOLLOW
 11 ;   THESE PATHS:
 12 ;                   KEYBOARD   TO   8086 COMMAND INTERPRETER
 13 ;                   KEYBOARD   TO   8086 EEPROM ROUTINE
 14 ;                   KEYBOARD   TO   MONITOR
 15 ;                   8086       TO   MONITOR
 16 ;                   EEPROM     TO   8086 COMMAND INTERPRETER
 17 ;                   EEPROM     TO   8086 EEPROM ROUTINE
 18 ;                   EEPROM     TO   MONITOR
 19 ;
 20 ;   COMMAND CODES ARE:
 21 ;K E
 22 ;- -                CNTRL-A          MONITOR INHIBIT
 23 ;- -                CNTRL-B          MONITOR UNINHIBIT
 24 ;- -                CNTRL-C          EEPROM INHIBIT
 25 ;- -                CNTRL-D          EEPROM UNINHIBIT
 26 ;- -                CNTRL-E          TURN ON EEPROM BUFFER
 27 ;- -                CNTRL-F          DISPLAY PAGE 0 SELECTED
 28 ;- -                CNTRL-G          DISPLAY PAGE 1 SELECTED
 29 ;0 X                CNTRL-H          BACKSPACE (CURSOR LEFT)
 30 ;X X                CNTRL-I          TAB (EVERY 8 CHARACTERS)
 31 ;X X                CNTRL-J          LINEFEED (CURSOR DOWN)
 32 ;X -                CNTRL-K          TURN EEPROM BUFFER OFF
 33 ;X X                CNTRL-L          ERASE PAGE
 34 ;X X                CNTRL-M          CARRIAGE RETURN
 35 ;X X                CNTRL-N          TURN OFF BACKGROUND/FOREGROUND*
 36 ;X X                CNTRL-O          TURN ON BACKGROUND/FOREGROUND*
 37 ;X X                CNTRL-P          SET COLOR TO BLACK
 38 ;X X                CNTRL-Q          SET COLOR TO RED
 39 ;X X                CNTRL-R          GREEN
 40 ;X X                CNTRL-S          YELLOW
 41 ;X X                CNTRL-T          BLUE
 42 ;X X                CNTRL-U          MAGENTA
 43 ;X X                CNTRL-V          CYAN
 44 ;X X                CNTRL-W          WHITE
 45 ;0 X                CNTRL-X          ABORT LINE
 46 ;X X                CNTRL-Y          MOVE CURSOR RIGHT
 47 ;X X                CNTRL-Z          MOVE CURSOR DOWN AND LEFT
 48 ;X X                CNTRL-^          MOVE CURSOR UP
 49 ;X X                CNTRL-\          HOME CURSOR
 50 ;- -                CNTRL-DEL        RECALL EEPROM BUFFER
 51 ;
```

```
LINE SOURCE
  52 ;
  53 ;   THE TWO COLUMNS ASSOCIATED WITH EACH CONTROL KEY REPRESENT TH/
       -E
  54 ;   APPROPRIATE KEYBOARD AND EEPROM BUFFER ACTION CONNECTED WITH /
       -THAT
  55 ;   KEY.
  56 ;                   -  KEYSTROKE NOT STORED IN BUFFER
  57 ;                   X  KEYSTROKE STORED IN BUFFER
  58 ;                   O  OPERATION PERFORMED ON BUFFER
  59 ;
  60 ;   A CHARACTER IS STORED IN THE EEPROM BUFFER ONLY IF THE OPERAT/
       -ION
  61 ;   WAS PERFORMED ON THE MONITOR.
  62 DUMBTERM          SEGMENT
  63 ;
  64 ;   8275 REGISTERS
  65 ;
  66 CRT_REGS          STRUC
  67    CRT_PARAM:     DW       1
  68    CRT_COM_STAT:  DW       1
  69 CRT_REGS          ENDS
  70 ;
  71 ;   8279 REGISTERS
  72 ;
  73 KYBD_REGS         STRUC
  74    KBD_DATA:      DW       1
  75    KBD_COM_STAT:  DW       1
  76 KYBD_REGS         ENDS
  77 ;
  78 ;   8086/8089 COMMON FLAGS
  79 ;
  80 DP_RAM_FLAGS      STRUC
  81    TP_LSW:        DW       1
  82    TP_MSD:        DW       1
  83    EEP_INH:       DB       1
  84    EEP_BUF_FULL:  DB       1
  85    EEP_RECALL:    DB       1
  86    COL_CH_INH:    DB       1
  87    KBD_INH:       DB       1
  88    KBD_BUF_FULL:  DB       1
  89 ;
  90 ;
  91    COM_8086:      DB       1
  92    COLOR:         DB       1
  93    STR_PTR_8086:  DW       1
  94    BACK_COL_SW:   DB       1
  95    MON_INH:       DB       1
  96    DSPLY_PG_PTR:  DB       1
  97    SCROLL_REQ:    DB       1
  98    NEW_CHAR_FLAG:         DB       1
  99    NEW_CHAR:              DB       1
 100 ;
 101 ;
 102    MON_HOM:       DW       1
 103    MON_END:       DW       1
 104    MON_LMARG:     DW       1
 105    MON_RMARG:     DW       1
```

```
LINE SOURCE
 106    KBD_BUF_PTR:    DW        1
 107    E2_MON_INH:     DB        1
 108 ;
 109 DP_RAM_FLAGS       ENDS
 110 ;
 111 ;   DISPLAY CHARACTER STRUCTURE
 112 ;
 113 CHAR_DEF           STRUC
 114    COLOR_MODE:     DB        1
 115    ASCII_GRAPH1:   DB        1
 116    GRAPH_2AND3:    DB        1
 117    GRAPH_4AND5:    DB        1
 118 CHAR_DEF           ENDS
 119 ;
 120 ;   PRIVATE 8089 FLAGS
 121 ;
 122 STAT_RAM_FLAGS     STRUC
 123    STACK:          DW        1
 124    STACK_MSD:      DW        1
 125                    DW        1
 126                    DW        1
 127    EEP_BUF_PTR:    DW        1
 128 ;
 129 ;
 130    LINE_CNT:       DW        1
 131    CHAR_CNT:       DW        1
 132 ;
 133 ;
 134    ASCII:          DB        1
 135    ASCII_TEMP:     DB        1
 136    CURSOR_X1:      DB        1
 137    CURSOR_X2:      DB        1
 138    CURSOR_Y1:      DB        1
 139    CURSOR_Y2:      DB        1
 140 ;
 141 ;
 142    LINE_TEMP:      DW        1
 143    CHAR_TEMP:      DW        1
 144    PAGE_INDEX:     DW        1
 145 STAT_RAM_FLAGS     ENDS
 146 ;
 147 ;   ADDRESS EQUATES
 148 ;
 149 STAT_RAM           EQU       00000H
 150 CRT1               EQU       06000H
 151 CRT2               EQU       04000H
 152 CLK_EN             EQU       08000H
 153 CRT_DATA           EQU       0A000H
 154 KYBD               EQU       0C000H
 155 ;
 156 ;
 157 DSPLY_PAGE0        EQU       0F8000H
 158 DSPLY_PAGE1        EQU       0FC000H
 159 COM_BUF            EQU       0FBD00H
 160 EEP_BUF            EQU       0FBE00H
 161 KEY_BUF            EQU       0FBF00H
 162 DP_PB              EQU       0FFF00H
```

```
LINE SOURCE
 163 ;
 164 ;   DATA/COMMAND EQUATES
 165 ;
 166 EOF             EQU     0FFH
 167 CRT_RST         EQU     000H
 168 CRT_PARAM1      EQU     04F4FH
 169 CRT_PARAM2      EQU     06F6FH
 170 CRT_PARAM3      EQU     04444H
 171 CRT_PARAM4      EQU     00606H
 172 CRT_CURSOR      EQU     08080H
 173 CRT_CNTR        EQU     0E0E0H
 174 START_DISP      EQU     02020H
 175 END_DISP_PG     EQU     15360
 176 KBD_STR_SET     EQU     006H
 177 KBD_PRG_CLK     EQU     034H
 178 KBD_FIFO_RD     EQU     050H
 179 ;*************************************************
 180 ;***********  INITIALIZATION  ********************
 181 ;*************************************************
 182 ;
 183 ;   TURN ON THE CRT CHARACTER CLOCK AND RESET THE
 184 ;   CRT CONTROLLERS
 185 ;
 186 START:
 187         MOVI    GB,CLK_EN
 188         MOVI    [GB],001H
 189         MOVI    GB,CRT1
 190         MOVI    GC,CRT2
 191         MOVI    [GC].CRT_COM_STAT,CRT_RST
 192         MOVI    [GB].CRT_COM_STAT,CRT_RST
 193 ;
 194 ;   SUPPLY THE FOUR PARAMETER BYTES THAT SPECIFY
 195 ;   80X48 CHARACTERS, TRANSPARENT ATTRIBUTES, AND
 196 ;   A BLINKING UNDERLINE CURSOR
 197 ;
 198         MOVI    [GB],CRT_PARAM1
 199         MOVI    [GB],CRT_PARAM2
 200         MOVI    [GB],CRT_PARAM3
 201         MOVI    [GB],CRT_PARAM4
 202         MOVI    [GC],CRT_PARAM1
 203         MOVI    [GC],CRT_PARAM2
 204         MOVI    [GC],CRT_PARAM3
 205         MOVI    [GC],CRT_PARAM4
 206 ;
 207 ;   SET CURSOR TO UPPER LEFT CORNER OF MONITOR
 208 ;
 209         MOVI    [GC].CRT_COM_STAT,CRT_CURSOR
 210         MOVI    [GC],000H
 211         MOVI    [GC],000H
 212         MOVI    [GB].CRT_COM_STAT,CRT_CURSOR
 213         MOVI    [GB],000H
 214         MOVI    [GB],000H
 215 ;
 216 ;   SYNCHRONIZE 8275 CLUSTER BY RESETTING COUNTERS
 217 ;
 218         MOVI    [GC].CRT_COM_STAT,CRT_CNTR
 219         MOVI    [GB].CRT_COM_STAT,CRT_CNTR
```

```
LINE SOURCE
 220          MOVI     GC,STAT_RAM
 221          MOVBI    [GC].CURSOR_X1,000H
 222          MOVBI    [GC].CURSOR_X2,000H
 223          MOVBI    [GC].CURSOR_Y1,000H
 224          MOVBI    [GC].CURSOR_Y2,000H
 225 ;
 226 ;   INITIALIZE 8279 KEYBOARD CONTROLLER
 227 ;
 228          MOVI     GB,KYBD
 229          MOVI     [GB].KBD_COM_STAT,KBD_STR_SET
 230          MOVI     [GB].KBD_COM_STAT,KBD_PRG_CLK
 231          MOVI     [GB].KBD_COM_STAT,KBD_FIFO_RD
 232 ;
 233 ;   INITIALIZE 8089 FLAGS
 234 ;
 235          MOVI     GC,STAT_RAM
 236          LPDI     GA,DP_PB
 237          MOVI     [GC].LINE_CNT,000H
 238          MOVI     [GC].CHAR_CNT,000H
 239 ;
 240 ;
 241          MOVBI    [GA].EEP_INH,0FFH
 242          MOVBI    [GA].EEP_BUF_FULL,00H
 243          MOVBI    [GA].EEP_RECALL,00H
 244          MOVBI    [GA].KBD_INH,00H
 245          MOVBI    [GA].KBD_BUF_FULL,00H
 246          MOVBI    [GA].COM_8086,00H
 247          MOVBI    [GA].COLOR,038H
 248          MOVBI    [GA].BACK_COL_SW,00H
 249          MOVBI    [GA].COL_CH_INH,00H
 250          MOVBI    [GA].SCROLL_REQ,00H
 251          MOVBI    [GA].DSPLY_PG_PTR,00H
 252          MOVBI    [GA].MON_INH,00H
 253          MOVBI    [GA].E2_MON_INH,0
 254          MOVI     [GA].MON_HOM,00H
 255          MOVI     [GA].MON_END,048
 256          MOVI     [GA].MON_RMARG,080
 257          MOVI     [GA].MON_LMARG,00H
 258 ;
 259 ;   INITIALIZE 8089 POINTER
 260 ;
 261          MOVI     [GC].EEP_BUF_PTR,00H
 262          MOVI     [GA].STR_PTR_8086,00H
 263          MOVI     [GA].KBD_BUF_PTR,000H
 264 ;************************************************
 265 ;***********  EXECUTIVE  ************************
 266 ;************************************************
 267 ;
 268 ;   DMA SET-UP
 269 ;
 270 ;   LOAD CHANNEL CONTROL REGISTER TO SPECIFY:
 271 ;                MEMORY TO PORT
 272 ;                SYNCHRO ON DEST
 273 ;                GA POINTS TO SOURCE
 274 ;                TERMINATE ON EXT
 275 ;                TERMINATION OFFSET=0
 276 ;
```

```
LINE SOURCE
 277            MOVI    GC,CLK_EN
 278            MOVI    [GC],00H            ;INHIBIT CHAR CLOCK
 279                                        ;ON 8275 TO SYNCHRONIZE
 280            MOVI    GC,CRT1
 281            MOVI    [GC].CRT_COM_STAT,START_DISP
 282            MOVI    GC,CRT2
 283            MOVI    [GC].CRT_COM_STAT,START_DISP
 284 DMA_LP:
 285            MOVI    CC,0512OH
 286 ;
 287 ;  SETUP DESTINATION AND THEN
 288 ;  SOURCE ACCORDING TO DISPLAY PAGE
 289 ;  POINTER
 290 ;
 291            MOVI    GB,CRT_DATA
 292            LPDI    GA,DSPLY_PAGE0
 293            LPDI    GC,DP_PB
 294            JZB     [GC].DSPLY_PG_PTR,SOURCE_OK
 295            LPDI    GA,DSPLY_PAGE1
 296 SOURCE_OK:
 297            JNZB    [GC].MON_INH,DMA_BYPASS ;IF THE MONITOR IS INHIB/
       -ITED
 298                                                ;BYPASS THE DMA
 299            JNZB    [GC].E2_MON_INH,DMA_BYPASS_1
 300            MOVI    GC,CLK_EN
 301 ;
 302 ;  START CRT CHARACTER CLOCK AND BEGIN DMA
 303 ;
 304            XFER
 305            MOVI    [GC],01H
 306            SINTR
 307 ;
 308 ;  SIGNAL THE 8086 THAT END OF FRAME HAS OCCURED AND THE UPDATIN/
       -G OF THE
 309 ;  INTERRUPT DRIVEN SECONDS COUNTER MAY BEGIN
 310 ;
 311 ;
 312 ;  READ CRT STATUS REGISTERS IN ORDER TO RESET IRQ
 313 ;
 314            MOVI    GC,CRT1
 315            MOV     GA,[GC].CRT_COM_STAT
 316            MOVI    GC,CRT2
 317            MOV     GB,[GC].CRT_COM_STAT
 318            JMP     DMA_BYPASS
 319 DMA_BYPASS_1:
 320            MOVI    GC,120
 321 E2_WAIT_LOOP:
 322            MOVI    GB,300
 323 E2_INNER_LOOP:
 324            DEC     GB
 325            JNZ     GB,E2_INNER_LOOP
 326            DEC     GC
 327            JNZ     GC,E2_WAIT_LOOP
 328 DMA_BYPASS:
 329 ;
 330 ;  CHECK FOR STRING FROM 8086
 331 ;  IT HAS PRIORITY OVER KEYBOARD
```

```
LINE SOURCE
 332 ;
 333          LPDI     GC,DP_PB
 334          JNZB     [GC].COM_8086,STRING_86
 335 ;
 336 ;   CHECK 8279 KYBD STATUS
 337 ;
 338          MOVI     GB,KYBD
 339          MOVB     GA,[GB].KBD_COM_STAT
 340          ANDI     GA,0FH
 341          LJNZ     GA,READ_KYBD     ;KEY DOWN
 342 ;
 343 ;   UPDATE THE CURSOR POSITION
 344 ;
 345 CURSOR_UPDATE:
 346          LPDI     GC,DP_PB
 347 ;
 348 ;   CHECK FOR 86 COMMAND CHARACTER MODE AND PROCESS
 349 ;   THE NEXT BYTE
 350          JZB      [GC].COM_8086,COM_STR_BYPASS
 351          INC      [GC].STR_PTR_8086
 352          JMP      GET_COM
 353 COM_STR_BYPASS:
 354          MOVI     GB,CRT1
 355          MOVI     GC,STAT_RAM
 356          MOVI     [GB].CRT_COM_STAT,CRT_CURSOR
 357          MOVB     GA,[GC].CHAR_CNT          ;SET UP FOR X POSITION
 358          MOVB     [GC].CURSOR_X1,GA         ;CURSOR OUTPUT
 359          MOVB     [GC].CURSOR_X2,GA         ;BY DOUBLING UP
 360          MOVB     GA,[GC].LINE_CNT
 361          MOVB     [GC].CURSOR_Y1,GA         ;SAME FOR Y POSITION
 362          MOVB     [GC].CURSOR_Y2,GA
 363          MOV      [GB],[GC].CURSOR_X1
 364          MOV      [GB],[GC].CURSOR_Y1
 365          MOVI     GB,CRT2                   ;DO IT FOR ALL
 366          MOVI     [GB].CRT_COM_STAT,CRT_CURSOR
 367          MOV      [GB],[GC].CURSOR_X1       ;CONTROLLERS
 368          MOV      [GB],[GC].CURSOR_Y1
 369 INTR_86:
 370          JMP      DMA_LP
 371 STRING_86:
 372          MOVI     [GC].STR_PTR_8086,00H
 373 GET_COM:
 374          MOV      IX,[GC].STR_PTR_8086
 375          LPDI     GB,COM_BUF
 376 ;
 377 ;   GET NEXT COMMAND CHARACTER FROM THE 8086
 378 ;   AND SAVE IT AS A KEYSTROKE
 379 ;
 380          MOVB     GA,[GB+IX]
 381          LPDI     GC,COM_BUF        ;***TEST CODE****
 382          MOVB     GA,[GB + IX]      ;***
 383          LPDI     GC,DP_PB          ;***
 384          MOVI     GB,STAT_RAM
 385          MOVB     [GB].ASCII,GA
 386 ;
 387 ;   CHECK FOR END OF COMMAND STRING
 388 ;
```

```
LINE SOURCE
 389          MOVI     MC,0FFFFH
 390          JMCNE    [GB].ASCII,COM_CNT
 391 ;
 392 ;   END OF COMMAND STRING-RESET COMMAND FLAG
 393 ;
 394          MOVBI    [GC].COM_8086,00H
 395          JMP      CURSOR_UPDATE
 396 READ_KYBD:
 397 ;
 398 ;   TEMPORARY GET CHAR ROUTINE
 399 ;
 400          JNZB     [GC].KBD_INH,CURSOR_UPDATE
 401          JNZB     [GC].KBD_BUF_FULL,CURSOR_UPDATE
 402 ;
 403 ;   IF THE KEYBOARD IS INHIBITED OR THE BUFFER FULL,
 404 ;   DONT READ THE 8279
 405 ;
 406          MOVB     GA,[GB].KBD_DATA
 407          NOT      GA
 408          ANDI     GA,007FH
 409          MOVB     [GC].NEW_CHAR,GA
 410          MOVBI    [GC].NEW_CHAR_FLAG,1
 411          MOVI     GB,STAT_RAM
 412          MOVB     [GB].ASCII,GA              ;SAVE KEYSTROKE
 413 COM_CNT:
 414          LPDI     GB,DP_PB
 415          MOVI     GC,STAT_RAM
 416 ;
 417 ;   CHECK FOR FIRST CHARACTER AFTER CNTRL-DEL, THIS CHARACTER WILL
 418 ;   BE PLACED IN EEP_RECALL AND USED FOR SELECTING WHICH EEP BUFF/
       -ER
 419 ;   IS TO BE RECALLED
 420 ;
 421          MOVB     GA,[GB].EEP_RECALL         ;IF MSB OF EEP_RECALL IS/
       - SET
 422          ANDI     GA,007FH                   ;USE PRESENT ASCII CHARA/
       -CTER
 423          JZ       GA,NO_RECALL               ;AS INDEX FOR EEPROM REC/
       -ALL
 424          MOVB     GA,[GC].ASCII
 425          MOVB     [GB].EEP_RECALL,GA
 426          JMP      CURSOR_UPDATE
 427 NO_RECALL:
 428 ;
 429 ;   CHECK FOR FIRST CHARACTER AFTER CNTRL_E
 430 ;   THIS CHARACTER WILL BE PLACED IN THE
 431 ;   EEPROM BUFFER AND NOT PROCESSED
 432 ;
 433          JNZB     [GB].EEP_INH,EEP_BYPASS
 434          JNZ      [GC].EEP_BUF_PTR,EEP_BYPASS
 435 ;
 436 ;   INSERT ASCII CHARACTER
 437 ;
 438          MOV      IX,[GC].EEP_BUF_PTR
 439          MOVB     GA,[GC].ASCII
 440          LPDI     GB,EEP_BUF
 441          MOVB     [GB+IX],GA
```

```
LINE SOURCE
 442          INC       [GC].EEP_BUF_PTR
 443          JMP       CURSOR_UPDATE
 444 EEP_BYPASS:
 445 ;
 446 ;   CHECK FOR NON CONTROL CHARACTER
 447 ;
 448          MOVI      MC,06000H
 449          LJMCNE    [GC].ASCII,CHAR_OUT
 450 ;
 451 ;*************************************************************
 452 ;*************** CONTROL KEY DECODE **************************
 453 ;*************************************************************
 454 ;
 455 ;   LOOK FOR 8086 COMMAND STRING SO CERTAIN
 456 ;   COMMANDS WILL NOT BE AVAILABLE FROM
 457 ;   KEYBOARD
 458 ;
 459 ;         JZB       [GB].COM_8086,NOT_CNTRLG
 460 ;
 461 ;   CHECK FOR MONITOR INHIBIT
 462 ;         (CNTRL-A)
 463 ;
 464          MOVI      MC,07F01H
 465          JMCNE     [GC].ASCII,NOT_CNTRLA
 466          MOVBI     [GB].MON_INH,0FFH
 467          JMP       CURSOR_UPDATE
 468 NOT_CNTRLA:
 469 ;
 470 ;   CHECK FOR MONITOR UNINHIBIT
 471 ;         (CNTRL-B)
 472 ;
 473          MOVI      MC,07F02H
 474          JMCNE     [GC].ASCII,NOT_CNTRLB
 475          MOVBI     [GB].MON_INH,00H
 476          JMP       CURSOR_UPDATE
 477 NOT_CNTRLB:
 478 NOT_CNTRLC:
 479 NOT_CNTRLD:
 480 ;
 481 ;   CHECK FOR SET DISPLAY PAGE 0
 482 ;         (CNTRL-F)
 483 ;
 484          MOVI      MC,07F06H
 485          JMCNE     [GC].ASCII,NOT_CNTRLF
 486          MOVBI     [GB].DSPLY_PG_PTR,00H
 487          JMP       CURSOR_UPDATE
 488 NOT_CNTRLF:
 489 ;
 490 ;   CHECK FOR SET DISPLAY PAGE 1
 491 ;         (CNTRL-G)
 492 ;
 493          MOVI      MC,07F07H
 494          JMCNE     [GC].ASCII,NOT_CNTRLG
 495          MOVBI     [GB].DSPLY_PG_PTR,0FFH
 496          JMP       CURSOR_UPDATE
 497 NOT_CNTRLG:
 498 ;
```

```
LINE SOURCE
 499 ;   THE FOLLOWING CONTROL COMMANDS ARE
 500 ;   AVAILABLE THROUGH THE 8089 KEYBOARD
 501 ;
 502 ;
 503 ;   LOOK FOR CARRIAGE RETURN
 504 ;
 505           MOVI     MC,07F0DH
 506           LJMCE    [GC].ASCII,CHAR_CR
 507 ;
 508 ;   LOOK FOR BACKSPACE
 509 ;
 510           MOVI     MC,07F08H
 511           LJMCE    [GC].ASCII,BACK_SPACE
 512 ;
 513 ;   LOOK FOR COLOR CONTROL KEYS
 514 ;   CNTRL-P THRU CNTRL-W
 515 ;
 516           MOVI     MC,07810H
 517           LJMCE    [GC].ASCII,COLOR_KEY
 518 ;
 519 ;   CHECK FOR SET BACKGROUND COLOR FLAG
 520 ;         (CNTRL-N)
 521 ;
 522           MOVI     MC,07F0EH
 523           LJMCE    [GC].ASCII,CNTRL_N
 524 ;
 525 ;   CHECK FOR SET FOREGROUND COLOR
 526 ;         (CNTRL-O)
 527           MOVI     MC,07F0FH
 528           LJMCE    [GC].ASCII,CNTRL_O
 529 ;
 530 ;
 531 ;
 532 ;   CHECK FOR EEPROM BUFFER RECALL
 533 ;         (CNTRL-DEL)
 534           MOVI     MC,07F1FH
 535           LJMCE    [GC].ASCII,EEP_DUMP
 536 ;
 537 ;   LOOK FOR TAB
 538 ;         (CNTRL-I)
 539 ;
 540           MOVI     MC,07F09H
 541           LJMCE    [GC].ASCII,CURSOR_TAB
 542 ;
 543 ;   LOOK FOR ERASE PAGE
 544 ;         (CNTRL-L)
 545 ;
 546           MOVI     MC,07F0CH
 547           LJMCE    [GC].ASCII,ERASE_PAGE
 548 ;
 549 ;   LOOK FOR CANCEL LINE
 550 ;         (CNTRL-X)
 551 ;
 552           MOVI     MC,07F18H
 553           LJMCE    [GC].ASCII,CNTRL_X
 554 ;
 555 ;   LOOK FOR HOME THE CURSOR
```

```
LINE SOURCE
 556 ;          (CNTRL \)
 557 ;
 558            MOVI     MC,07F1CH
 559            LJMCE    [GC].ASCII,CURSOR_HOME
 560
 561 ;   LOOK FOR UP CURSOR
 562 ;          (CNTRL ^)
 563            MOVI     MC,07F1EH
 564            LJMCE    [GC].ASCII,UP_CURSOR
 565 ;
 566 ;   LOOK FOR DOWN CURSOR
 567 ;          (CNTRL J)
 568 ;
 569            MOVI     MC,07F0AH
 570            LJMCE    [GC].ASCII,DWN_CURSOR
 571 ;
 572 ;   LOOK FOR RIGHT CURSOR
 573 ;          (CNTRL-Y)
 574 ;
 575            MOVI     MC,07F19H
 576            LJMCE    [GC].ASCII,RIGHT_CURSOR
 577 ;
 578 ;   LOOK FOR DOWN AND LEFT CURSOR
 579 ;          (CNTRL-Z)
 580 ;
 581            MOVI     MC,07F1AH
 582            LJMCE    [GC].ASCII,BACK_DOWN
 583 ;
 584 ;   ALL OTHER KEY INPUTS ARE IGNORED
 585 ;
 586            JMP      CURSOR_UPDATE
 587 ;*************************************************
 588 ;*************  CONTROL SEGMENTS  ****************
 589 ;*************************************************
 590 ;
 591 ;
 592 ;   SET THE COLOR BACKGROUND/FOREGROUND* FLAG TO
 593 ;   BACKGROUND (0)
 594 ;
 595 CNTRL_N:
 596            MOVI     GB,STAT_RAM
 597            LPDI     GC,DP_PB
 598 ;
 599 ;   CHECK FOR MONITOR OR COLOR CHANGE INHIBITED
 600 ;
 601            JNZB     [GC].COL_CH_INH,KEEP_BF
 602            MOVBI    [GC].BACK_COL_SW,00H
 603 KEEP_BF:
 604            LJMP     KEY_EEP_EXIT
 605 ;
 606 ;   SET THE COLOR BACKGROUND/FOREGROUND* FLAG
 607 ;   TO FOREGROUND
 608 ;
 609 CNTRL_0:
 610            MOVI     GB,STAT_RAM
 611            LPDI     GC,DP_PB
 612 ;
```

```
LINE SOURCE
 613 ;  CHECK FOR MONITOR OR COLOR CHANGE INHIBITED
 614 ;
 615          JNZB     [GC].COL_CH_INH,KEEP_BF2
 616          MOVBI    [GC].BACK_COL_SW,OFFH
 617 KEEP_BF2:
 618          LJMP     KEY_EEP_EXIT
 619 ;
 620 ;  TURN ON THE EEPROM BUFFER
 621 ;       (CNTRL_E)
 622 ;
 623 ;  THIS ROUTINE INITIALIZES THE EEPROM BUFFER
 624 ;  POINTER
 625 ;
 626 CNTRL_E:
 627          MOVI     GB,STAT_RAM
 628          LPDI     GC,DP_PB
 629          LJNZB    [GC].EEP_BUF_FULL,CURSOR_UPDATE
 630          MOVBI    [GC].EEP_BUF_FULL,00H    ;********
 631          MOVI     [GB].EEP_BUF_PTR,00H
 632          MOVBI    [GC].EEP_INH,00H
 633          JMP      CURSOR_UPDATE
 634 ;
 635 ;  TURN THE EEPROM BUFFER OFF
 636 ;
 637 CNTRL_K:
 638          MOVI     GB,STAT_RAM
 639          LPDI     GC,DP_PB
 640          LCALL    [GB],KEY_BUF_UPDATE
 641          MOVBI    [GC].EEP_BUF_FULL,OFFH
 642          MOVBI    [GC].EEP_INH,OFFH
 643          MOV      IX,[GB].EEP_BUF_PTR
 644          LPDI     GA,EEP_BUF
 645 ;
 646 ;  INSERT END OF FILE MARKER
 647 ;
 648          MOVBI    [GA+IX],OFFH
 649          INC      [GB].EEP_BUF_PTR
 650          JMP      CURSOR_UPDATE
 651 ;
 652 ;  DUMP EEPROM BUFFER 0-9
 653 ;
 654 EEP_DUMP:
 655          MOVI     GB,STAT_RAM
 656          LPDI     GC,DP_PB
 657          LPDI     GC,DP_PB
 658          MOVBI    [GC].EEP_RECALL,OFFH     ;SET FLAG TO ALL ONES, B/
      -UT IT
 659                                            ;WILL BE REPLACED BY THE/
      - NEXT
 660                                            ;ASCII CHARACTER
 661 ED_XIT:
 662          JMP      INTR_86
 663 CHAR_OUT:
 664          MOVI     GB,STAT_RAM
 665          LCALL    [GB],CHAR_TO_MON
 666 ;
 667 ;  PASS KEYSTROKES TO 8086
```

```
LINE SOURCE
 668 ;
 669 KEY_EEP_EXIT:
 670           MOVI      GB,STAT_RAM
 671           LCALL     [GB],KEY_BUF_UPDATE
 672 EEP_UP_EXIT:
 673           MOVI      GB,STAT_RAM
 674           LCALL     [GB],EEP_BUF_UPDATE
 675           JMP       CURSOR_UPDATE
 676 CHAR_CR:
 677           MOVI      GB,STAT_RAM
 678           LCALL     [GB],CR_UPDATE
 679 ;
 680 ;   SET KEYBOARD AND EEPROM BUFFER FULL
 681 ;   FLAGS IF NOT INHIBITED
 682 ;
 683           MOVI      GB,STAT_RAM
 684           LPDI      GC,DP_PB
 685           JNZB      [GC].COM_8086,CURSOR_UPDATE      ;IF IN 8086 COMM/
       -AND
 686                                                      ;MODE, DONT ALTER
 687                                                      ;KEYBOARD STATUS
 688           MOVI      GB,STAT_RAM
 689           LCALL     [GB],KEY_BUF_UPDATE
 690           MOVBI     [GC].KBD_BUF_FULL,OFFH   ;******
 691 EEP_CHK:
 692           JMP       EEP_UP_EXIT
 693 ;
 694 ;   ALTER BACKGROUND OR FOREGROUND COLOR ACCORDING
 695 ;   TO THE 3 LEAST SIGNIFICANT BITS OF THE INPUT
 696 ;   KEY AND THE STATUS OF THE BACKGROUND/FOREGROUND*
 697 ;   FLAG.
 698 ;
 699 COLOR_KEY:
 700           MOVI      GB,STAT_RAM
 701           LPDI      GC,DP_PB
 702           LCALL     [GB],EEP_BUF_UPDATE
 703           LCALL     [GB],KEY_BUF_UPDATE
 704           LJNZB     [GC].COL_CH_INH,CURSOR_UPDATE
 705           MOVB      GA,[GC].BACK_COL_SW
 706 ;
 707 ;   CHECK B/F* FLAG
 708 ;
 709           JNZ       GA,BACK_GROUND
 710           MOVB      GA,[GB].ASCII
 711           ANDI      GA,07H
 712           MOV       [GB].ASCII,GA
 713           MOVB      GA,[GC].COLOR
 714           ANDI      GA,038H
 715 ;
 716 ;   OR INPUT COLOR INTO FOREGROUND SECTION OF COLOR BYTE
 717 ;
 718           ORB       GA,[GB].ASCII
 719           MOVB      [GC].COLOR,GA
 720           JMP       CURSOR_UPDATE
 721 BACK_GROUND:
 722           MOVB      GA,[GB].ASCII
 723           ADD       GA,[GB].ASCII
```

```
LINE SOURCE
 724            ADD       GA,[GB].ASCII
 725            ADD       GA,[GB].ASCII
 726            MOVB      [GB].ASCII_TEMP,GA
 727            ADD       GA,[GB].ASCII/TEMP
 728 ;
 729 ;   SHIFT INPUT COLOR OVER AND OR IT INTO THE BACKGROUND
 730 ;   SECTION OF THE COLOR BYTE
 731 ;
 732            ANDI      GA,038H
 733            MOV       [GB].ASCII,GA
 734            MOVB      GA,[GC].COLOR
 735            ANDI      GA,047H
 736            ORB       GA,[GB].ASCII
 737            MOVB      [GC].COLOR,GA
 738            JMP       CURSOR_UPDATE
 739 ;
 740 ;   TAB ROUTINE
 741 ;
 742 ;   THIS ROUTINE MOVES THE CURSOR TO THE NEXT
 743 ;   COLUMN WHOSE NUMBER IS A MULTIPLE OF 8.
 744 ;
 745 CURSOR_TAB:
 746            MOVI      GB,STAT_RAM
 747            LCALL     [GB],EEP_BUF_UPDATE
 748            LCALL     [GB],KEY_BUF_UPDATE
 749            LPDI      GC,DP_PB
 750 ;
 751 ;   CHECK FOR CHARACTER COUNT BEING A
 752 ;   MULTIPLE OF EIGHT (3 LSB = 0)
 753 ;
 754 TAB_CNT:
 755 ;
 756 ;   PLACE BLANK ON THE SCREEN
 757 ;
 758            MOVBI     [GB].ASCII,020H
 759            LCALL     [GB],CHAR_TO_MON
 760            MOV       GA,[GB].CHAR_CNT
 761            ANDI      GA,07H
 762            LJZ       GA,CURSOR_UPDATE
 763            JZB       [GC].SCROLL_REQ,TAB_CNT
 764            JMP       CURSOR_UPDATE
 765 ;
 766 ;   ERASE PAGE ROUTINE
 767 ;
 768 ;   THIS ROUTINE ERASES THE PAGE FROM THE CURRENT
 769 ;   CURSOR POSITION.   IT ENDS WITH THE CURSOR AT
 770 ;   THE HOME POSITION.
 771 ;
 772 ;
 773 ;   UP CURSOR ROUTINE
 774 ;
 775 UP_CURSOR:
 776            MOVI      GB,STAT_RAM
 777            LPDI      GC,DP_PB
 778            MOV       IX,[GC].MON_HOM
 779            NOT       IX                          ;CHECK FOR UPPER BOUNDARY
 780            AND       IX,[GB].LINE_CNT
```

```
LINE SOURCE
 781            LJZ       IX,CURSOR_UPDATE
 782            DEC       [GB].LINE_CNT
 783            JMP       KEY_EEP_EXIT
 784 ;
 785 ;   LINE FEED (DOWN CURSOR)
 786 ;
 787 DWN_CURSOR:
 788            MOVI      GB,STAT_RAM
 789            LPDI      GC,DP_PB
 790            MOV       IX,[GB].LINE_CNT          ;COMPARE PRESENT LINE
 791            INC       IX                        ;COUNT + 1 TO BOTTOM
 792            NOT       IX                        ;MARGIN
 793            AND       IX,[GC].MON_END           ;IF EQUAL ABORT CURSOR M/
         -OVE
 794            LJZ       IX,CURSOR_UPDATE
 795            INC       [GB].LINE_CNT             ;MOVE OK
 796            JMP       KEY_EEP_EXIT
 797 ;
 798 ;   MOVE CURSOR RIGHT
 799 ;
 800 RIGHT_CURSOR:
 801            MOVI      GB,STAT_RAM
 802            LPDI      GC,DP_PB
 803            MOV       IX,[GB].CHAR_CNT          ;COMPARE PRESENT CHARACT/
         -ER
 804            INC       IX                        ;COUNT + 1 TO RIGHT
 805            NOT       IX                        ;MARGIN
 806            AND       IX,[GC].MON_RMARG         ;IF EQUAL ABORT
 807            LJZ       IX,CURSOR_UPDATE          ;CURSOR MOVE
 808            INC       [GB].CHAR_CNT             ;MOV OK
 809            JMP       KEY_EEP_EXIT
 810 BACK_DOWN:
 811            MOVI      GB,STAT_RAM
 812            LPDI      GC,DP_PB
 813            MOV       IX,[GB].LINE_CNT          ;COMPARE PRESENT LINE
 814            INC       IX                        ;COUNT + 1 TO BOTTOM
 815            NOT       IX                        ;MARGIN
 816            AND       IX,[GC].MON_END           ;IF EQUAL ABORT CURSOR M/
         -OVE
 817            LJZ       IX,CURSOR_UPDATE
 818            MOV       IX,[GC].MON_LMARG         ;IF CURSOR IS AT LEFT MA/
         -RGIN
 819            NOT       IX                        ;ABORT CURSOR MOVE
 820            AND       IX,[GB].CHAR_CNT
 821            LJZ       IX,CURSOR_UPDATE
 822            INC       [GB].LINE_CNT
 823            DEC       [GB].CHAR_CNT
 824            JMP       KEY_EEP_EXIT
 825 ;
 826 ;   CANCEL THE PRESENT LINE
 827 ;
 828 CNTRL_X:
 829            MOVI      GB,STAT_RAM
 830            LPDI      GC,DP_PB
 831            MOV       [GB].CHAR_CNT,[GC].MON_LMARG
 832 ;
 833 ;   RESET THE KEYBOARD BUFFER POINTER
```

```
LINE SOURCE
 834 ;
 835            MOVBI    [GC].KBD_BUF_FULL,00H
 836            MOVI     [GC].KBD_BUF_PTR,00H
 837            JMP      KEY_EEP_EXIT
 838 ERASE_PAGE:
 839            MOVI     GB,STAT_RAM
 840            LCALL    [GB],EEP_BUF_UPDATE
 841            LCALL    [GB],KEY_BUF_UPDATE
 842            LPDI     GC,DP_PB
 843 ;
 844 ;   STORE BLANKS ON THE SCREEN
 845 ;
 846            MOVBI    [GB].ASCII,020H
 847 ERASE_CNT:
 848            LCALL    [GB],CHAR_TO_MON
 849            JZB      [GC].SCROLL_REQ,ERASE_CNT
 850            JMP      CH_NTR
 851 ;
 852 ;   HOME THE CURSOR
 853 ;
 854 CURSOR_HOME:
 855            MOVI     GB,STAT_RAM
 856            LCALL    [GB],EEP_BUF_UPDATE
 857            LCALL    [GB],KEY_BUF_UPDATE
 858 CH_NTR:
 859            LPDI     GC,DP_PB
 860            MOVBI    [GC].KBD_INH,00H
 861            MOVBI    [GC].SCROLL_REQ,00H
 862            MOV      [GB].CHAR_CNT,[GC].MON_LMARG
 863            MOV      [GB].LINE_CNT,[GC].MON_HOM
 864            JMP      CURSOR_UPDATE
 865 ;
 866 ;   PERFORM BACK-SPACE BY DECREMENTING THE DISPLAY
 867 ;   PAGE POINTER, KEYBOARD POINTER, EEPROM POINTER,
 868 ;   AND CURSOR POSITION
 869 ;
 870 BACK_SPACE:
 871            MOVI     GB,STAT_RAM
 872            LPDI     GC,DP_PB
 873            MOV      IX,[GC].MON_LMARG        ;IF CURSOR IS AT LEFT
 874            NOT      IX                       ;MARGIN ABORT BACKSPACE
 875            AND      IX,[GB].CHAR_CNT
 876            LJZ      IX,CURSOR_UPDATE
 877            DEC      [GB].CHAR_CNT
 878 ;
 879 ;   DO BACKSPACE IF MONITOR NOT INHIBITED AND CURSOR IS
 880 ;   NOT AT THE BEGINNING OF A LINE
 881 ;
 882 KYBD_UPDATE:
 883          LJNZB    [GC].KBD_BUF_FULL,EEP_EXIT
 884 ;
 885 ;   IF KEY BUFFER POINTER IS ZERO, DONT BACKSPACE IT
 886 ;
 887            JZ       [GC].KBD_BUF_PTR,EEP_EXIT
 888            DEC      [GC].KBD_BUF_PTR
 889 EEP_EXIT:
 890            MOVI     GB,STAT_RAM
```

```
LINE SOURCE
 891          JMP      EEP_UP_EXIT
 892 ;*****************************************************
 893 ;***************  SUBROUTINES  ***********************
 894 ;*****************************************************
 895 CHAR_TO_MON:
 896 ;
 897 ;   SET UP DISPLAY PAGE POINTER AND INDEX
 898 ;
 899          LPDI     GB,DSPLY_PAGE0
 900          LPDI     GC,DP_PB
 901          JZ       [GC].DSPLY_PG_PTR,PTR_OK
 902          LPDI     GB,DSPLY_PAGE1
 903 ;
 904 ;   COMPUTE 80XLINE_CNT
 905 ;
 906 PTR_OK:
 907          MOVI     GC,STAT_RAM
 908          MOV      GA,[GC].LINE_CNT
 909          ADD      GA,[GC].LINE_CNT
 910          ADD      GA,[GC].LINE_CNT
 911          ADD      GA,[GC].LINE_CNT
 912          ADD      GA,[GC].LINE_CNT
 913          MOV      [GC].LINE_TEMP,GA
 914          ADD      GA,[GC].LINE_TEMP        ; 2 X 5
 915          MOV      [GC].LINE_TEMP,GA
 916          ADD      GA,[GC].LINE_TEMP        ; 4 X 5
 917          MOV      [GC].LINE_TEMP,GA
 918          ADD      GA,[GC].LINE_TEMP        ; 8 X 5
 919          MOV      [GC].LINE_TEMP,GA
 920          ADD      GA,[GC].LINE_TEMP        ; 16 X 5
 921 ;
 922 ;   MEMORY POINTER = DISPLAY PAGE POINTER +
 923 ;                    4X(80XLINE_CNT + CHAR_CNT)
 924 ;
 925          ADD      GA,[GC].CHAR_CNT
 926          MOV      [GC].LINE_TEMP,GA
 927          ADD      GA,[GC].LINE_TEMP
 928          ADD      GA,[GC].LINE_TEMP
 929          ADD      GA,[GC].LINE_TEMP
 930          MOV      [GC].PAGE_INDEX,GA
 931          ADD      GB,[GC].PAGE_INDEX
 932 ;
 933 ;   SAVE ASCII CODE IN DISPLAY PAGE
 934 ;
 935          MOVB     [GB].ASCII_GRAPH1,[GC].ASCII
 936 ;
 937 ;   SAVE BACKGROUND AND FOREGROUND COLOR IN
 938 ;   DISPLAY PAGE
 939 ;
 940          LPDI     GC,DP_PB
 941          MOVB     [GB].COLOR_MODE,[GC].COLOR
 942 ;
 943 ;   CLEAR OTHER 2 DISPLAY PAGE BYTES
 944 ;
 945          MOVBI    [GB].GRAPH_2AND3,00H
 946          MOVBI    [GB].GRAPH_4AND5,00H
 947 ;
```

```
LINE SOURCE
 948 ;   INCREMENT X CURSOR POSITION AND CHARACTER POINTER.
 949 ;   CHECK FOR RIGHT MARGIN OVERRUN
 950 ;
 951           MOVI      GB,STAT_RAM
 952           INC       [GB].CHAR_CNT
 953           MOV       [GB].CHAR_TEMP,[GB].CHAR_CNT
 954           NOT       [GB].CHAR_TEMP
 955           MOV       GA,[GC].MON_RMARG
 956           AND       GA,[GB].CHAR_TEMP
 957           JNZ       GA,MON_UPDATE_FIN
 958 CR_UPDATE:
 959 ;   IF RIGHT MARGIN WAS EXCEEDED, MOVE CHARACTER COUNT
 960 ;   TO LEFT MARGIN AND INCREMENT LINE COUNT AND Y CURSOR
 961 ;   POSITION
 962           LPDI      GC,DP_PB
 963           MOVI      GB,STAT_RAM
 964           INC       [GB].LINE_CNT
 965           MOV       [GB].CHAR_CNT,[GC].MON_LMARG
 966 ;
 967 ;   CHECK IF LINE COUNT WENT PAST BOTTOM OF SCREEN
 968 ;
 969           MOV       [GB].LINE_TEMP,[GB].LINE_CNT
 970           NOT       [GB].LINE_TEMP
 971           MOV       GA,[GB].LINE_TEMP
 972           AND       GA,[GC].MON_END
 973           JNZ       GA,MON_UPDATE_FIN
 974 ;
 975 ;   LINE COUNT EXCEEDED BOTTOM MARGIN--
 976 ;   SET SCROLL FLAG
 977 ;   AND KEYBOARD INHIBIT AND DECREMENT LINE COUNT
 978 ;
 979           MOVBI     [GC].SCROLL_REQ,0FFH
 980           MOVBI     [GC].KBD_INH,0FFH ;****
 981           DEC       [GB].LINE_CNT
 982 MON_UPDATE_FIN:
 983 ;
 984 ;   RETURN TO CALLING ROUTINE
 985 ;
 986           MOVI      GB,STAT_RAM
 987           LPDI      GC,DP_PB
 988           MOVP      TP,[GB]
 989 ;
 990 ;   KEYBOARD BUFFER SUBROUTINE
 991 ;
 992 ;   TRANSFER THE ASCII CHARACTERS OBTAINED FROM THE
 993 ;   8279 CONTROLLER INTO A BUFFER FOR LATER
 994 ;   PROCESSING BY THE 8086.
 995 ;
 996 KEY_BUF_UPDATE:
 997           LPDI      GC,DP_PB
 998           MOVI      GB,STAT_RAM
 999 ;
1000 ;   BYPASS IF BUFFER FULL
1001 ;
1002           JNZB      [GC].KBD_BUF_FULL,KBU_RETURN
1003 ;
1004 ;   BYPASS IF 8086 COMMAND MODE
```

```
LINE SOURCE
1005 ;
1006          JNZB      [GC].COM_8086,KBU_RETURN
1007 ;
1008 ;   XFER THE CHARACTER
1009 ;
1010          MOV       IX,[GC].KBD_BUF_PTR
1011          LPDI      GA,KEY_BUF
1012          MOVB      [GA+IX],[GB].ASCII
1013          INC       [GC].KBD_BUF_PTR
1014          MOV       GA,[GC].KBD_BUF_PTR
1015          ANDI      GA,OFFOOH
1016          JZ        GA,KBU_RETURN
1017 ;
1018 ;   POINTER OVERRUN-SET BUFFER FULL FLAG
1019 ;
1020          DEC       [GC].KBD_BUF_PTR
1021          MOVBI     [GC].KBD_BUF_FULL,OFFH
1022          MOVBI     [GA+IX],OFFH              ;SET END OF BUFFER MARKER
1023 KBU_RETURN:
1024          MOVP      TP,[GB]
1025 ;
1026 ;   EEPROM BUFFER SUBROUTINE
1027 ;
1028 ;   THIS ROUTINE TRANSFERS THE ASCII CHARACTERS OBTAINED
1029 ;   FROM THE 8279 CONTROLLER INTO THE DUAL PORT EEPROM BUFFER
1030 ;
1031 EEP_BUF_UPDATE:
1032          MOVI      GB,STAT_RAM
1033          LPDI      GC,DP_PB
1034 ;
1035 ;   CHECK FOR BUFFER FULL FLAG OR EEPROM INHIBITED
1036 ;
1037          JNZB      [GC].EEP_INH,EBU_RETURN
1038          JNZB      [GC].EEP_BUF_FULL,EBU_RETURN
1039 ;
1040 ;   XFER THE CHARACTER
1041 ;
1042          MOV       IX,[GB].EEP_BUF_PTR
1043          LPDI      GA,EEP_BUF
1044          MOVB      [GA+IX],[GB].ASCII
1045          INC       [GB].EEP_BUF_PTR
1046          MOV       GA,[GB].EEP_BUF_PTR
1047          ANDI      GA,OFFOOH
1048          JZ        GA,EBU_RETURN
1049 ;
1050 ;   POINTER OVERRUN-SET BUFFER FULL FLAG
1051 ;
1052          DEC       [GB].EEP_BUF_PTR
1053          MOVBI     [GC].EEP_BUF_FULL,OFFH
1054 EBU_RETURN:
1055          MOVP      TP,[GB]
1056 DUMBTERM          ENDS
1057                   END
```